



尽享5小时完整视频教程！跟着数十万人的Python导师学Python！

PEARSON

“笨办法”

学

Python

Learn

PYTHON

the **HARD WAY**

THIRD EDITION

(第3版)

[美] Zed A. Shaw 著
王巍巍 译



人民邮电出版社
POSTS & TELECOM PRESS

目 录

[封面](#)

[扉页](#)

[版权](#)

[版权声明](#)

[译者序](#)

[前言](#)

[习题0 准备工作](#)

[Mac OSX](#)

[OSX: 应该看到的结果](#)

[Windows](#)

[Windows: 应该看到的结果](#)

[Linux](#)

[Linux: 应该看到的结果](#)

[给新手的告诫](#)

[习题1 第一个程序](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题2 注释和#号](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题3 数字和数学计算](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题4 变量和命名](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题5 更多的变量和打印](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题6 字符串和文本](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题7 更多打印](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题8 打印，打印](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题9 打印，打印，打印](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题10 那是什么](#)

[应该看到的结果](#)

[转义序列](#)

[附加练习](#)

[常见问题回答](#)

[习题11 提问](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题12 提示别人](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题13 参数、解包和变量](#)

[等一下！“特性”还有另外一个名字](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题14 提示和传递](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题15 读取文件](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题16 读写文件](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题17 更多文件操作](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题18 命名、变量、代码和函数](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题19 函数和变量](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题20 函数和文件](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题21 函数可以返回某些东西](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题22 到现在你学到了哪些东西](#)

[学到的东西](#)

[习题23 阅读一些代码](#)

[习题24 更多练习](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题25 更多更多的实践](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题26 恭喜你，现在可以考试了！](#)

[常见问题回答](#)

[习题27 记住逻辑关系](#)

[逻辑术语](#)

[真值表](#)

[常见问题回答](#)

[习题28 布尔表达式练习](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题29 if语句](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题30 else和if](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题31 作出决定](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题32 循环和列表](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题33 while循环](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题34 访问列表的元素](#)

[附加练习](#)

[习题35 分支和函数](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题36 设计和调试](#)

[if语句的规则](#)

[循环的规则](#)

[调试的小技巧](#)

[家庭作业](#)

[习题37 复习各种符号](#)

[关键字](#)

[数据类型](#)

[字符串转义序列](#)

[字符串格式化](#)

[操作符](#)

[阅读代码](#)

[附加练习](#)

[常见问题回答](#)

[习题38 列表的操作](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题39 字典，可爱的字典](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题40 模块、类和对象](#)

[模块和字典差不多](#)

[类和模块差不多](#)

[对象相当于迷你导入](#)

[获取某样东西里包含的东西](#)

[第一个关于类的例子](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题41 学习面向对象术语](#)

[单词练习](#)

[语汇练习](#)

[混合巩固练习](#)

[阅读测试](#)

[练习从语言到代码](#)

[阅读更多代码](#)

[常见问题回答](#)

[习题42 对象、类及从属关系](#)

[代码写成什么样子](#)

[关于class Name\(object\)](#)

[附加练习](#)

[常见问题回答](#)

[习题43 基本的面向对象分析和设计](#)

[简单游戏引擎的分析](#)

[把问题写下来或者画出来](#)

[摘录和研究关键概念](#)

[为各种概念创建类层次结构图和对象关系图](#)

[编写和运行各个类](#)

[重复和优化](#)

[自顶向下与自底向上](#)

[《来自Percal 25号行星的哥顿人》的代码](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题44 继承与合成](#)

[什么是继承](#)

[隐式继承](#)

[显式覆盖](#)

[在运行前或运行后替换](#)

[三种方式组合使用](#)

[为什么要用super\(\)](#)

[super\(\)和__init__搭配使用](#)

[合成](#)

[继承和合成的应用场合](#)

[附加练习](#)

[常见问题回答](#)

[习题45 你来制作一个游戏](#)

[评价你的游戏](#)

[函数的风格](#)

[类的风格](#)

[代码风格](#)

[好的注释](#)

[为你的游戏评分](#)

[习题46 项目骨架](#)

[Python软件包的安装](#)

[创建骨架项目目录](#)

[最终目录结构](#)

[测试你的配置](#)

[使用这个骨架](#)

[小测验](#)

[常见问题回答](#)

[习题47 自动化测试](#)

[编写测试用例](#)

[测试指南](#)

[应该看到的结果](#)

[附加练习](#)

[常见问题回答](#)

[习题48 更复杂的用户输入](#)

[我们的游戏语汇](#)

[断句](#)

[语汇元组](#)

[扫描输入](#)

[异常和数字](#)

[应该测试的东西](#)

[设计提示](#)

[附加练习](#)

[常见问题回答](#)

[习题49 创建句子](#)

[match和peek](#)

[句子的文法](#)

[关于异常](#)

[应该测试的东西](#)

[附加练习](#)

[常见问题回答](#)

[习题50 你的第一个网站](#)

[安装lpthw.web](#)

[写一个简单的“Hello World”项目](#)

[会发生什么](#)

[修正错误](#)

[创建基本的模板文件](#)

[附加练习](#)

[常见问题回答](#)

[习题51 从浏览器中获取输入](#)

[Web的工作原理](#)

[表单的工作原理](#)

[创建HTML表单](#)

[创建布局模板](#)

[为表单撰写自动测试代码](#)

[附加练习](#)

[常见问题回答](#)

[习题52 创建Web游戏](#)

[重构习题43中的游戏](#)

[会话和用户跟踪](#)

[创建引擎](#)

[期末考试](#)

[常见问题回答](#)

[接下来的路](#)

[怎样学习任何一种编程语言](#)

[老程序员的建议](#)

附录 命令行快速入门

简介：废话少说，命令行来也

如何使用这个附录

你需要发挥记忆力

习题1 准备工作

任务

知识点

更多任务

习题2 路径、文件夹和目录（pwd）

任务

知识点

更多任务

习题3 如果你迷失了

任务

知识点

习题4 创建目录（mkdir）

任务

知识点

更多任务

习题5 更改目录（cd）

任务

知识点

更多任务

习题6 列出目录下的内容（ls）

任务

知识点

更多任务

习题7 删除路径（rmdir）

[任务](#)

[知识点](#)

[更多任务](#)

[习题8 在多个目录中切换（pushd, popd）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题9 创建空文件（touch, New-Item）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题10 复制文件（cp）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题11 移动文件（mv）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题12 查看文件内容（less, MORE）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题13 流文件内容显示（cat）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题14 删除文件（rm）](#)

[任务](#)

[知识点](#)

[更多任务](#)

[习题15 退出命令行 \(exit\)](#)

[任务](#)

[知识点](#)

[更多任务](#)

[命令行将来的路](#)

[Unix Bash参考资料](#)

[PowerShell参考资料](#)

PEARSON

“笨办法”学Python（第3版）

Learn PYTHON the HARD WAY THIRD EDITION

[美]Zed A.Shaw 著

王巍巍 译

人民邮电出版社

北京

图书在版编目（CIP）数据

“笨办法”学Python：第3版/（美）肖（Shaw,Z.A.）著；王巍巍译.--
北京：人民邮电出版社，2014.11

ISBN 978-7-115-35054-1

I.①笨... II.①肖...②王... III.①软件工具—程序设计
IV.①TP311.56

中国版本图书馆CIP数据核字（2014）第078421号

内容提要

本书是一本Python入门书籍，适合对计算机了解不多，没有学过编程，但对编程感兴趣的读者学习使用。这本书以习题的方式引导读者一步一步学习编程，从简单的打印一直讲到完整项目的实现，让初学者从基础的编程技术入手，最终体验到软件开发的基本过程。

本书结构非常简单，共包括52个习题，其中26个覆盖了输入/输出、变量和函数三个主题，另外26个覆盖了一些比较高级的话题，入条件判断、循环、类和对象、代码测试及项目的实现等。每一章的格式基本相同，以代码习题开始，按照说明编写代码，运行并检查结果，然后再做附加练习。

◆著 [美]Zed A.Shaw

译 王巍巍

责任编辑 杨海玲

责任印制 彭志环 焦志炜

◆人民邮电出版社出版发行 北京市丰台区成寿寺路11号

邮编 100164 电子邮件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

北京鑫正大印刷有限公司印刷

◆开本：800×1000 1/16

印张：16.5

字数：373千字 2014年11月第1版

印数：1-3000册 2014年11月北京第1次印刷

著作权合同登记号 图字：01-2013-8978号

定价：49.00元（附光盘）

读者服务热线：**(010)81055410** 印装质量热线：**(010)81055316**

反盗版热线：**(010)81055315**

广告经营许可证：京崇工商广字第**0021**号

版权声明

Authorized translation from the English language edition,entitled Learn Python the Hard Way,Third Edition,9780321884916 by Zed A.Shaw,published by Pearson Education,Inc.,publishing as Addison-Wesley,Copyright © 2014 by Pearson Education,Inc.

All rights reserved.No part of this book may be reproduced or transmitted in any form or by any means,electronic or mechanical,including photocopying,recording or by any information storage retrieval system,without permission from Pearson Education,Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD.and POSTS & TELECOM PRESS Copyright © 2014.

本书中文简体字版由Pearson Education Asia Ltd.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

译者序

2010年的某一天，忽然在网上某处看到一个叫Zed A.Shaw的人在写一本Python入门书，而且这本书和别的入门书不太一样，于是就去了解了一下。虽然当时我也算是入过门了，但温故知新总是不错的。Zed的书是放在网上用Git做版本管理的，于是每更新一章我就跟着看一章。这段时间，除了巩固了一些Python知识，还亲眼见识了一本书是怎样写成的，真是收获不少。

看完后最大的一个感想就是：“原来入门书可以这么写！”

2011年年初正好我杂事较少，于是征得Zed的同意，也学着他的方式，在网上建了一个项目，目的是将这本书翻译为中文版。经过几位朋友的转发，这个项目也算得到了一定的关注度。断断续续3个月后，翻译初稿就算完成了，放在网上供大家随意阅览。也许到现在你还能找到网上流传的版本。

很可能 Zed 也没想到自己的书会获得这么高的关注度，有一次他在网上说：“如果我能在每个初学者身上赚一块钱，那我就差不多发财了。”后来Zed自费出版了这本书，甚至还开过线上和线下的教学课程，书的内容也在各种反馈的基础上逐步修改和完善。到2013年出第3版时，章节的深度和主题的覆盖度和当初已经差别很大了。

在这个阶段，我也一直试图让自己的翻译版本和Zed的更新版本内容保持一致。直到有一天，Zed忽然撤下了在线的Git仓库，由于我的项目没法和他的同步，翻译也无法进行下去了。经过沟通才知道，原来是大名鼎鼎的Addison-Wesley要出版他的书，由于版权问题，不只是他的

Git仓库，连我的翻译版也要下线。

虽然挺沮丧的，但我还是把刚开工翻译的第3版下线了。

正好这段时间人民邮电出版社的编辑联系到我，说是有机会出版这本书，于是我又高兴起来啦。

截至目前，本书已经发行到了第3版。第1版只是基本的编程入门，第2版重写了若干章节，加入了面向对象以及Web应用开发等相关的内容，第3版中作者根据学生反馈，在各章节添加了“常见问题回答”，并且再次修订了后面的若干重点章节，进一步加强了面向对象编程的部分。

感谢

翻译本书的动力来自于李飞林，如果没有他的鼓励，我很可能就半途而废了。在翻译这本书的过程中收到过大量热心网友的问题反馈和改进建议。人民邮电出版社勤劳而又专业的编辑们审稿和校对让这本书的文字显得不那么业余。在此一并谢谢。

前言

这本书的目的是让你起步编程。虽然书名说是用“Hard Way”（笨办法）学习写程序，但其实并非如此。所谓的“笨办法”指的是本书的教学方式，也就是所谓的“指令式”教学。在这个过程中，我会让你完成一系列习题，而你则通过重复练习来学到技能，这些习题也是专为重复练习而设计的。对于一无所知的初学者来说，在能理解更复杂的话题之前，这种教授方式效果是很好的。你可以在各种场合看到这种教授方式，从武术到音乐不一而足，甚至在学习基本的算术和阅读技能时也会看到这种教学方式。

这本书通过练习和记忆的方式，教你逐渐掌握Python的技能，然后由浅入深，让你将这些技能应用到各种问题上。读完本书以后，你将有能力接触更为复杂的编程主题。我喜欢告诉别人，我的这本书能给你一个“编程黑带”，意思就是说，你已经打好了基础，可以真正开始学习编程了。

如果你肯努力，并投入一些时间，学会了这些技能，你将学会如何编写代码。

致谢

首先我要感谢在本书前两版中帮过我的Angela，没有她的话我有可能就不会费工夫完成这两本书了。她帮我修订了第1版草稿，而且在我写书的过程中给了我极大的支持。

我还要感谢Greg Newman为前两版提供的封面设计，Brian Shumate在早期网站设计方面的帮助，以及所有读过前两版并且提出反馈和纠错

的读者。

谢谢你们。

笨办法更简单

在这本书的帮助下，你将通过完成下面这些非常简单的事情来学会一门编程语言，这也是每个程序员的必经之路。

- 1.从头到尾完成每一个习题。
- 2.一字不差地录入每一段程序。
- 3.让程序运行起来。

就是这样了。刚开始这对你来说会非常难，但你需要坚持下去。如果你通读了这本书，每晚花一两个小时做做习题，你可以为自己读下一本编程书籍打下良好的基础。通过这本书，你学到的可能不是真正的“编程”技术，但你会学到学习一门编程语言的基本技能。

这本书的目的是教会你编程新手所需的三种最重要的技能：读和写、注重细节以及发现不同。

读和写

很显然，如果你连打字都成问题的话，那你学习编程也会成问题。尤其是，如果你连程序源代码中的那些奇怪字符都打不出来的话，就更别提编程了。如果没有这些基本技能，你将连最基本的软件工作原理都难以学会。

手动录入代码范例并让它们运行起来的过程，会让你学会各种符号的名称，熟悉它们的用处，最终读懂编程语言。

注重细节

区分好程序员和差程序员的最重要的一个方面就是对于细节的重视程度。事实上这是任何行业区分好坏的标准。如果缺乏对于工作中每一个微小细节的注意，你的工作成果将不可避免地出现各种关键缺陷。从编程这一行来讲，你得到的结果将会是毛病多多而且难以使用的软件。

通读这本书并一字不差地录入书中的每个例子，会训练你把精力集

中到作品的细节上。

发现不同

程序员长年累月的工作会培养出一种重要的技能，那就是观察事物间不同点的能力。有经验的程序员拿着两份仅有细微不同的程序，可以立即指出里边的不同点来。程序员甚至制造出工具来让这件事更加容易，不过我们不会用到这些工具。你要先用笨办法训练自己，然后才可以使用这些工具。

在做这些习题并且录入代码的时候，你一定会写错东西，这是不可避免的，即使有经验的程序员也会偶尔出错。你的任务是把自己写的东西和正确答案对比，把所有的不同点都修正过来。这样的过程可以让你对程序里的错误和bug更加敏感。

不要复制粘贴

你必须手动将每个习题录进去。复制粘贴会让这些习题变得毫无意义。这些习题的目的是训练你的双手和大脑思维，让你有能力读代码、写代码、观察代码。如果你复制粘贴的话，就是在欺骗自己，而且这些习题的效果也会大打折扣。

关于坚持练习的一点提示

你通过这本书学习编程时，我正在学习弹吉他。我每天至少训练2小时，至少花1小时练习音阶、和弦、琶音，剩下的时间用来学习音乐理论和乐曲演奏、训练听力等。有时我一天会花8小时来学习吉他和音乐，因为我觉得这是一件有趣的事情。对我来说，要学习一样东西，最自然、最根本的方法就是去反复地练习。我知道，要学好一种技能，每日的练习是必不可少的，就算哪天的练习没啥进展（对我来说是常事），或者说学习内容实在太难，你也不必介意。只要坚持尝试，总有一天困难会变得容易，枯燥也会变得有趣。

通过这本书学习编程的过程中要记住一点，就是所谓的“万事开头难”，对于有价值的事情尤其如此。也许你是一个害怕失败的人，一遇

到困难就想放弃；也许你是一直没学会自律，一遇到“无聊”的事情就不想上手；也许因为有人夸你“有天赋”而让你自视甚高，不愿意做这些看上去很笨拙的事情，怕有负你“神童”的称号；也许你太过激进，把自己跟像我这样有20多年经验的编程老手相比，让自己失去了信心。

不管是什么原因，你一定要坚持下去。如果遇到做不出来的附加练习，或者遇到一个看不懂的习题，你可以暂时跳过去，过一阵子回来再看。编程中有一件经常发生的怪事就是，一开始你什么都不懂，这会让你感觉很不舒服，就像学习人类的自然语言一样，你会发现很难记住一些词语和特殊符号的用法，而且会经常感到很迷茫，直到有一天，忽然一下子你会觉得豁然开朗，以前不明白的东西忽然就明白了。如果你坚持完成并努力理解这些习题，你最终会学会这些东西的。也许你不会成为一位编程大师，但你至少会明白编程的原理。

如果你放弃的话，你会失去达到这个程度的机会。如果你坚持尝试，坚持录习题，坚持弄懂习题的话，你最终一定会明白里边的内容的。

如果你通读了这本书，却还是不懂怎样写代码，你的努力也不会白费。你可以说你已经尽力了，虽然成效不佳，至少你尝试过了。这也是一件值得骄傲的事情。

给“小聪明”们的警告

有些学过编程的人读到这本书可能会有一种被贬低的感觉。其实本书中没有任何要居高临下地贬低任何人的意思，只不过我比我面向的读者群知道的更多而已。如果你觉得自己比我聪明，觉得我在居高临下，那我也没办法，因为你根本就不是我的目标读者。

如果你觉得这本书里到处都在贬低你的智商，那我对你有以下三个建议。

- 1.别读这本书了。我这本书不是写给你的，而是写给那些不是什么都懂的人看的。

2.放下架子好好学。如果你认为你什么都懂，那就很难从比自己强的人身上学到什么了。

3.学Lisp去。我听说什么都懂的人特喜欢Lisp。

对于其他抱着学习的目的而来的人，你们读的时候就想着我在微笑就可以了，而且我的眼睛里还带点儿恶作剧的闪光。

习题0 准备工作

这个习题并没有代码内容，它的主要目的是让你在计算机上安装好 Python。你应该尽量照着说明进行操作，例如，Mac OSX 默认已经安装了 Python 2，所以就不要在上面安装 Python 3 或者别的 Python 版本了。

注意 如果你不知道怎样使用 Windows 下的 PowerShell，或者 OSX 下的 Terminal（终端），或者 Linux 下的 Bash，那你就需要先学会一个。我把我写的一本《命令行快速入门》简化了一下放到了本书的附录里，读完那部分后，再回来继续下面的步骤。

Mac OSX

完成这个习题你需要完成下列任务。

- 1.用浏览器打开<http://www.barebones.com/products/textwrangler/>找到并安装TextWrangler文本编辑器。
- 2.把TextWrangler（也就是你的编辑器）放到Dock中，以方便日后使用。
- 3.找到系统中的Terminal程序。到处找找，你会找到的。
- 4.把Terminal也放到Dock里面。
- 5.运行Terminal程序，这个程序没什么好看的。
- 6.在Terminal里运行python。运行的方法是输入程序的名字再敲一下回车键。
- 7.按Ctrl+D（^D）退出python。
- 8.这样你就应该退回到敲python前的提示界面了。如果没有的话，自己研究一下为什么。
- 9.学着在Terminal上创建一个目录。
- 10.学着在Terminal上变到一个目录。
- 11.使用你的编辑器在你进入的目录下建立一个文件。建立一个文件，使用“保存”（Save）或者“另存为”（Save As...）选项，然后选择这个目录。
- 12.使用键盘切换回到Terminal窗口。
- 13.回到Terminal，看看你能不能使用命令看到你新建的文件，上网搜索如何将文件夹中的内容列出来。

OSX: 应该看到的结果

下面是我在自己电脑的Terminal中完成上述步骤时看到的内容，和你做的结果会有一些不同，看看你能不能找出两者的不同点。

```
Last login: Sat Apr 24 00:56:54 on ttys001
```

```
~ $ python
```

```
Python 2.5.1 (r251:54863, Feb 6 2009, 19:02:12)
```

```
[GCC 4.0.1 (Apple Inc.build 5465)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> ^D
```

```
~ $ mkdir mystuff
```

```
~ $ cd mystuff
```

```
mystuff $ ls
```

```
# ...使用TextWrangler编辑test.txt ...
```

```
mystuff $ ls
```

```
test.txt
```

```
mystuff $
```

Windows

1.用浏览器打开 <http://notepad-plus-plus.org> 下载并安装 Notepad++文本编辑器。这个操作无需管理员权限。

2.把Notepad++放到桌面或者快速启动栏，这样就可以方便地访问该程序了。这两条在安装选项中可以看到。

3.从开始菜单运行PowerShell程序。你可以使用开始菜单的搜索功能，输入名称后敲回车键即可运行。

4.为它创建一个快捷方式，放到桌面或者快速启动栏中以方便使用。

5.运行终端程序（也就是PowerShell），这个程序没什么好看的。

6.在终端程序中运行python。运行的方法是输入程序的名字再敲一下回车键。

a.如果运行python发现它不存在（python is not recognized），你需要访问<http://python.org/download>下载并且安装Python。

b.确认你要安装的是Python 2而不是Python 3。

c.你也可以试试ActiveState Python，尤其是没有管理员权限的时候。

d.如果安装好了，但是python还是不能被识别，那你需要在PowerShell下输入并执行以下命令：

```
[Environment]::SetEnvironmentVariable("Path",  
"$env:Path;C:\Python27", "User")
```

e.关闭并重启PowerShell，确认python现在可以运行。如果不行的话，可能需要重启电脑。

- 7.按Ctrl+Z (^Z) 退出python。
 - 8.这样你就应该退回到敲python前的提示界面了。如果没有的话，自己研究一下为什么。
 - 9.学着在终端创建一个目录。
 - 10.学着在终端上变到一个目录。
 - 11.使用你的编辑器在你进入的目录下建立一个文件。建立一个文件，使用“保存”（Save）或者“另存为”（Save As...）选项，然后选择这个目录。
 - 12.使用键盘切换回到终端窗口。
 - 13.回到终端，看看你能不能使用命令看到你新建的文件。
- 注意 Windows下安装的Python可能默认没有正确配置路径。确认你在PowerShell下输入了[Environment]::SetEnvironmentVariable("Path", "\$env:Path;C:\Python27","User")。你也许需要重启PowerShell或者计算机来让路径设置生效。

Windows: 应该看到的结果

```
> python
ActivePython 2.6.5.12 (ActiveState Software Inc.) based on
Python 2.6.5 (r265:79063, Mar 20 2010, 14:22:52) [MSC v.1500 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ^Z

> mkdir mystuff
> cd mystuff
...使用Notepad++编辑mystuff目录下的test.txt...
>
```

<如果你没有使用管理员权限安装，你会看到一堆错误，忽略它们，按回车即可。>

> dir

Volume in drive C is

Volume Serial Number is 085C-7E02

Directory of C:\Documents and Settings\you\mystuff

04.05.2010	23:32	<DIR>	.
04.05.2010	23:32	<DIR>	..
04.05.2010	23:32		6 test.txt
		1 File(s)	6 bytes
		2 Dir(s)	14 804 623 360 bytes free

>

你看到的命令行提示、Python信息及其他一些东西可能会非常不一样，这只是一个大致的情況罢了。

Linux

Linux 系统可谓五花八门，安装软件的方式也各有不同。既然你是Linux 用户，我就假设你已经知道如何安装软件包了，以下是操作说明。

- 1.使用你的Linux软件包管理器并安装gedit文本编辑器。
- 2.把gedit（也就是你的编辑器）放到窗口管理器显见的位置，以方便日后使用。
 - a.运行gedit，先改掉一些愚蠢的默认设定。
 - b.从gedit菜单中打开Preferences，选择Editor页面。
 - c.将Tab width:改为4。
 - d.选择（确认已勾选该选项）Insert spaces instead of tabs。
 - e.然后打开Automatic indentation选项。
 - f.转到View选项卡，打开Display line numbers选项。
- 3.找到Terminal程序。它的名字可能是GNOME Terminal、Konsole或者xterm，以下均以Terminal代称。
- 4.把Terminal也放到你的Dock里面。
- 5.运行Terminal程序，这个程序没什么好看的。
- 6.在Terminal程序中运行python。运行的方法是输入程序的名字再敲一下回车键。（如果运行python发现它不存在，你需要安装它，而且要确认你安装的是Python 2而非Python 3。）
- 7.按Ctrl+D（^D）退出python。
- 8.这样你就应该退回到敲python前的提示界面了。如果没有的话，自己研究一下为什么。

- 9.学着在Terminal上创建一个目录。
- 10.学着在Terminal上变到一个目录。
- 11.使用你的编辑器在你进入的目录下建立一个文件。建立一个文件，使用“保存”（Save）或者“另存为”（Save As...）选项，然后选择这个目录。
- 12.使用键盘切换回到Terminal窗口，如果不知道怎样使用键盘切换你可以自己查一下。
- 13.回到Terminal，看看你能不能使用命令列出你新建的文件。

Linux: 应该看到的结果

```
$ python
Python 2.6.5 (r265:79063, Apr 1 2010, 05:28:39)
[GCC 4.4.3 20100316 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>

$ mkdir mystuff
$ cd mystuff
# ...使用gedit编辑text.txt ...
$ ls
test.txt
$
```

你看到的命令行提示、Python信息及其他一些东西可能会非常不一样，这只是一个大致的情況罢了。

给新手的告诫

你已经完成了这个习题。取决于对计算机的熟悉程度，这个习题对你而言可能会有些难。如果你觉得有难度的话，你要自己克服困难，多花点时间学习一下，因为如果你不会这些基础操作的话，编程对你来说将会更难学。

如果有程序员告诉你使用vim或者emacs，你就对他们说“不”。当你水平达到一定程度的时候，这些编辑器才适合你用。你现在需要的只是一个可以编辑文本的编辑器。我们使用gedit、TextWrangler、Notepad++是因为它们很简单，而且在不同的系统上面使用起来都是一样的。就连专业程序员也会使用这些文本编辑器，所以对你而言，用它们入门编程已经足够了。

也许有程序员会告诉你安装和学习Python 3。你应该告诉他们“等你电脑里的所有Python代码都是Python 3的了，我再试着学学吧。”你这句话足够他们忙活十来年了。

总有一天你会听到有程序员建议你使用Mac OSX或者Linux。如果他喜欢字体美观，他会告诉你弄台 Mac OSX 计算机，如果他们喜欢操作控制而且留了一脸大胡子，他会让你安装Linux。这里再次向你说明，只要是一台手上能用的计算机就可以了。你需要的只有三样东西：一个文本编辑器，一个命令行终端，还有Python。

最后要说的是，这个习题的准备工作的目的就是让你可以在以后的习题中顺利地做到下面几件事。

1. 撰写习题的代码，在 Linux 下用 gedit，OSX 下用 TextWrangler，Windows 下用Notepad++。

2.运行你写的习题代码。

3.代码终端的时候修正错误的地方。

4.重复上述步骤。

其他的事情只会让你更困惑，所以还是坚持按计划进行吧。

习题1 第一个程序

你应该在习题 0 上花了不少的时间，学会了如何安装文本编辑器，运行文本编辑器，以及如何运行终端。如果你还没有完成这些练习，请不要继续往下进行了，否则你不会觉得很好过的。写在习题开头警告你不要跳过前面内容的警示本书中仅此一次，切记切记。

将下面的内容写到一个文件中，取名为`ex1.py`。这种命名方式很重要，Python文件最好以`.py`结尾。

`ex1.py`

```
1  print "Hello World!"
2  print "Hello Again"
3  print "I like typing this."
4  print "This is fun."
5  print 'Yay! Printing.'
6  print "I'd much rather you 'not'."
7  print 'I "said" do not touch this.'
```

如果你使用的是Mac OSX下的TextWrangler，那你的文本编辑器大致是图1-1所示的这个样子。

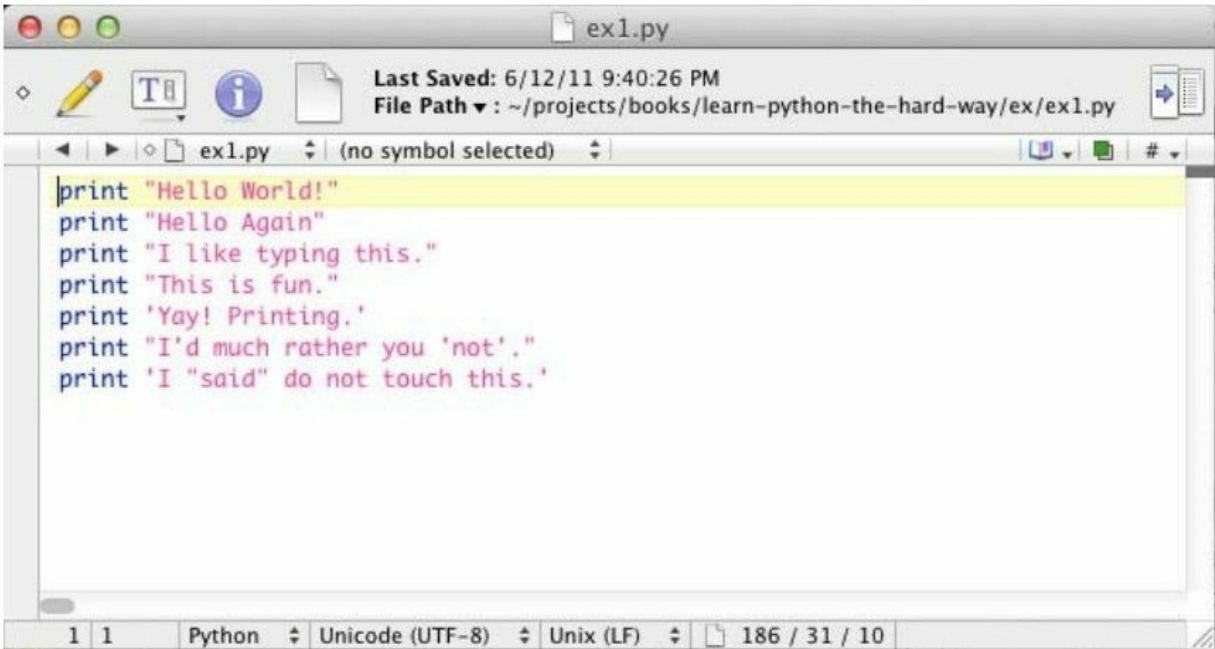


图1-1

如果你是在Windows下使用Notepad++, 那你看到的应该是图1-2所示的这个样子。

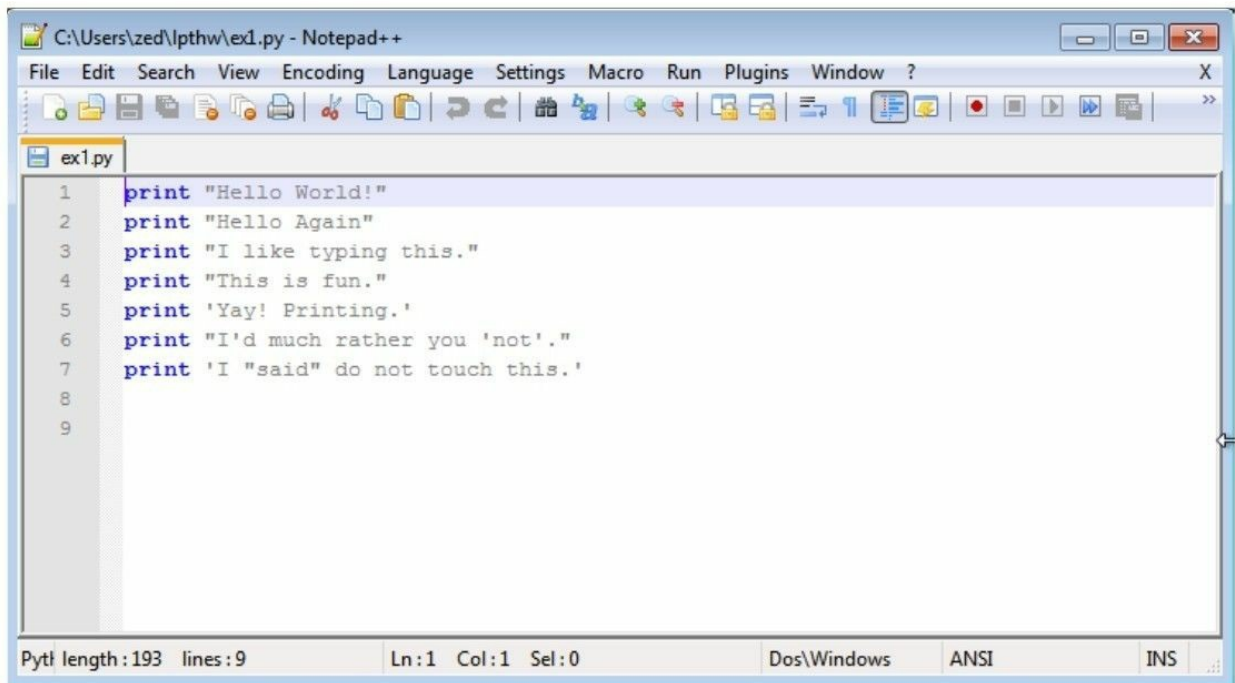


图1-2

别担心编辑器长得是不是一样，关键是以下几点。

1.注意我没有键入左边的行号。这些是额外加到书里边的，以便对代码具体的某一行进行讨论。例如“参见第5行.....”你无需将这些也写进Python脚本中去。

2.注意截图中开始的print语句，它和代码范例中是完全一样的。这里要求你做到“完全相同”，仅做到“差不多相同”是不够的。要让这段脚本正常工作，代码中的每个字符都必须完全匹配。当然，你的编辑器显示的颜色可能不一样，这并不重要，只有你键入的字符才是重要的。

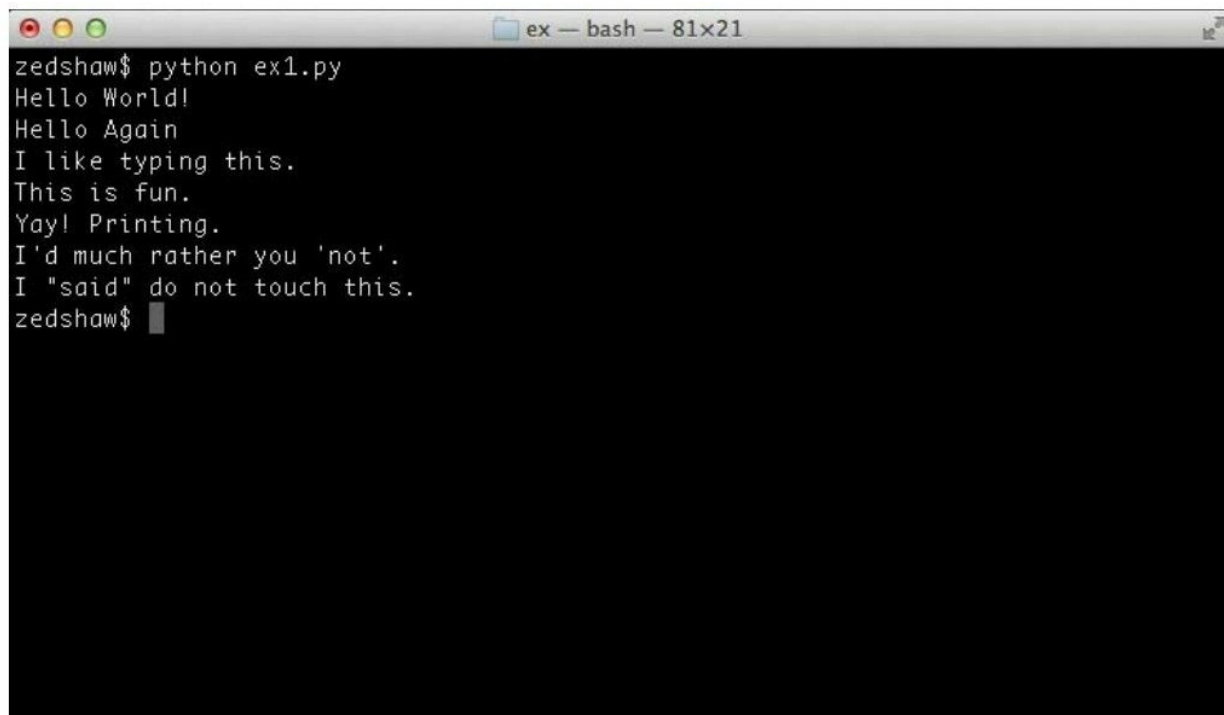
然后需要在终端通过输入以下内容来运行这段代码：

```
python ex1.py
```

如果你写对了，你应该看到和下面一样的内容。如果不一样，就是你哪儿弄错了。不是计算机出错了，计算机不会错。

应该看到的结果

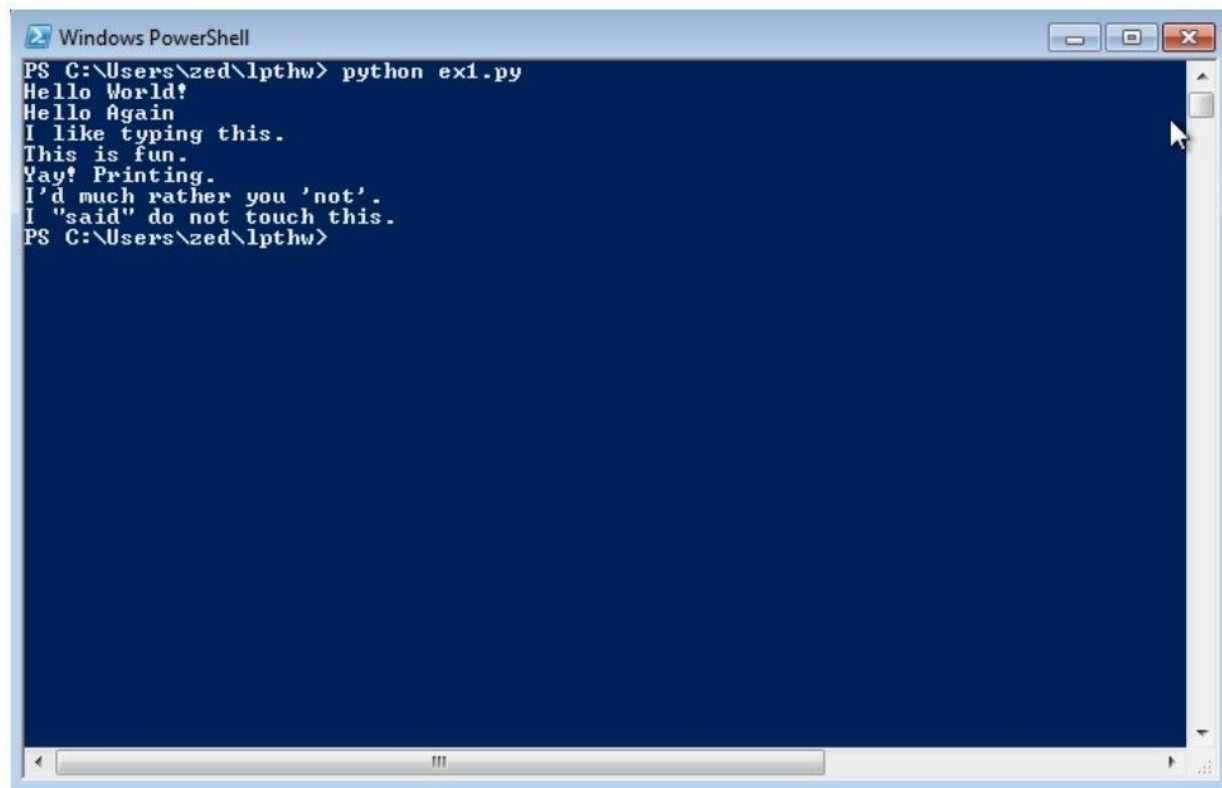
在Mac OSX的终端下面应该看到图1-3所示的这个样子。

A screenshot of a Mac OSX terminal window. The window has a title bar with three colored buttons (red, yellow, green) on the left and a title "ex — bash — 81x21" in the center. The terminal content shows a user prompt "zedshaw\$" followed by the command "python ex1.py". The output of the script is displayed line by line: "Hello World!", "Hello Again", "I like typing this.", "This is fun.", "Yay! Printing.", "I'd much rather you 'not'.", and "I \"said\" do not touch this.". The prompt "zedshaw\$" is shown again at the bottom with a cursor.

```
zedshaw$ python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
zedshaw$
```

图1-3

在Windows的PowerShell下应该看到图1-4所示的这个样子。



```
PS C:\Users\zed\lpthw> python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
PS C:\Users\zed\lpthw>
```

图1-4

你也许会看到`python ex1.py`前面显示的用户名、计算机名及其他一些信息不一样，这不是问题，重要的是你键入了命令，而且看到了相同的输出。

如果有错误，你会看到与下面类似的错误信息：

```
$ python ex/ex1.py
File "ex/ex1.py", line 3
    print "I like typing this.
    ^
```

SyntaxError: EOL while scanning string literal

你应该学会看懂这些内容，这是很重要的一点，因为你以后还会犯类似的错误。就是现在的我也会犯这样的错误。让我们一行一行来看。

- 1.首先我们在终端输入命令来运行`ex1.py`脚本。
- 2.Python告诉我们`ex1.py`文件的第3行有一个错误。

3.然后这一行的内容被显示出来。

4.然后Python显示一个插入符（^）符号，用来指示出错的位置。注意到少了一个双引号（"）了吗？

5.最后，它显示一个“语法错误”（SyntaxError），告诉你究竟是什么样的错误。通常这些错误信息都非常难懂，不过你可以把错误信息的内容复制到搜索引擎里，然后你就能看到别人也遇到过这样的错误，而且你也许能找到如何解决这个问题。

注意 如果你来自另外一个国家，而且你看到关于ASCII编码的错误，那就在你的Python脚本的最上面加入下面这一行：

```
# -*- coding: utf-8 -*-
```

这样你就在脚本中使用了Unicode UTF-8编码，这些错误就不会出现了。

附加练习

每个习题都有附加练习要完成。附加练习里边的内容是供你尝试的。如果你觉得做不出来，可以暂时跳过，过段时间再回来做。

在这个习题中，试试下面几件事儿。

- 1.让你的脚本再多打印一行。
- 2.让你的脚本只打印一行。
- 3.在一行的起始位置放一个“#”字符。它的作用是什么？自己研究一下。

从现在开始，除非特殊情况，我将不再解释每个习题的工作原理了。

注意 #（octothorpe）有很多的英文名字，如pound（英镑符）、hash（电话的#键）、mesh（网）等。选一个你觉得酷的就行了。

常见问题回答

这些是本书在线的时候收到的真实的学生问题，其中的一些问题你也有可能遇到，所以我就把这些问题以及它们的答案都搜集在这里了。

我可不可以使用**IDLE**？

不行。你应该使用OSX的终端或者Windows的PowerShell，和我这里演示的一样。如果你不知道如何用它们，可以去阅读附录的“命令行快速入门”。

怎样让编辑器显示不同颜色？

编辑之前先将文件保存为.py格式，如ex1.py，后面编辑时你就可以看到各种颜色了。

运行**ex1.py**时看到**SyntaxError: invalid syntax**。

你也许已经运行了Python，然后又在Python环境下运行了一遍Python。关掉并重启终端，重来一遍，只键入python ex1.py就可以了。

遇到错误信息**can't open file 'ex1.py': [Errno 2] No such file or directory**。

你需要在自己创建文件的目录下运行命令。确保你事先使用 `cd` 命令进入了这层目录下。假如你的文件存在 `lpthw/ex1.py` 下面，那你需要先执行 `cd lpthw/`再运行 `python ex1.py`，如果你不明白该命令的意思，那就去看看第一个问题中提到的“命令行快速入门”吧。

怎样在代码中输入我们国家的语言文字？

确认在文件开头加入了这行：`# -*- coding: utf-8 -*-`。

我的文件无法运行，它直接回到了提示符，没有任何输出。

很有可能是你把代码做了字面理解，认为`print "Hello World!"`就是让你在文件中打印"Hello World!"，于是你没有输入 `print`。你的代码应该和我的一模一样。我的每行里边都有`print`，你的也要确保都有，这样代码才能正常运行。

习题2 注释和#号

程序里的注释是很重要的。它们可以用自然语言告诉你某段代码的功能是什么。想要临时移除一段代码时，你还可以用注释的方式临时禁用这段代码。这个习题就是让你学会在Python中注释。

ex2.py

```
1  # A comment, this is so you can read your program later.
2  # Anything after the # is ignored by python.
3
4  print "I could have code like this." # and the comment after is
ignored
5
6  # You can also use a comment to "disable" or comment out a piece
of code:
7  # print "This won't run."
8
9  print "This will run."
```

从现在开始，我将用这样的方式来演示代码。我一直在强调“完全相同”，不过你也不必按照字面意思理解。你的程序在屏幕上的显示可能会有些不同，重要的是你在文本编辑器中输入的文本的正确性。事实上，我可以用任何编辑器写出这段程序，而且内容是完全一样的。

应该看到的结果

习题2 会话

```
$ python ex2.py
```

I could have code like this.

This will run.

再说明一次，我不会再贴各种屏幕截图了。你应该明白上面的内容是输出内容的字面翻译，而\$ python ...下面的内容才是你应该关心的。

附加练习

- 1.弄清楚#字符的作用，而且记住它的名字（英文为octothorpe或者pound character）。
- 2.打开ex2.py文件，从后往前逐行检查。从最后一行开始，倒着逐个单词检查回去。
- 3.有没有发现什么错误呢？有的话就改正过来。
- 4.朗读你写的习题，把每个字符都读出来。有没有发现更多的错误呢？有的话也一样改正过来。

常见问题回答

你确定#字符的名称是**pound character**?

我叫它 octothorpe, 这个名字没有哪个国家用作别的意思, 而且所有的人都能看懂它的意思。每个国家都觉得他们的叫法最正确、最闪亮。对我来说这是自大狂的想法, 而且说真的, 与其去关心这种细枝末节, 还不如把时间花在更重要的事情上面, 比如好好学习编程。

如果#是注释的意思, 那么为什么# **-*- coding: utf-8 -*-**能起作用呢?

Python其实还是没把这行当做代码处理, 这种用法只是让字符编码格式被识别的一个取巧的方案, 或者说是一个没办法的办法吧。在编辑器设置里你还能看到一种类似的注释。

为什么**print "Hi # there."**里的#没被忽略掉?

这行代码里的#处于字符串内部, 所以它就是引号结束前的字符串中的一部分, 这时它只是一个普通字符, 而不代表注释的意思。

怎样做多行注释?

每行前面放一个#就可以了。

我们国家的键盘上找不到#字符, 怎么办?

有的国家要通过Alt键组合才能输入这个字符。你可以用搜索引擎找一下解决方案。

为什么要让我倒着阅读代码?

这样可以避免让你的大脑跟着每一段代码内容的意思走, 这样可以让你精确处理每个片段, 从而让你更容易发现代码中的错误。这是一个很好使的查错技巧。

习题3 数字和数学计算

每一种编程语言都包含处理数字和进行数学计算的方法。不必担心，程序员经常谎称他们是多么牛的数学天才，其实他们根本不是。如果他们真是数学天才，他们早就去从事数学相关的行业了，而不是写写广告程序和社交网络游戏，偷偷赚点小钱而已。

这个习题里有很多数学运算符号。我们来看一遍它们都叫什么名字。你要一边写一边念出它们的名字来，直到你念烦了为止。名字如下：

+ 加号

- 减号

/ 斜杠

* 星号

% 百分号

< 小于号

> 大于号

<= 小于等于号

>= 大于等于号

有没有注意到以上只是些符号，没有给出具体的运算操作呢？写完下面的练习代码后，再回到上面的列表，写出每个符号的作用。例如，+是用来做加法运算的。

ex3.py

```
1 print "I will now count my chickens:"
```

```
2
3  print "Hens", 25 + 30 / 6
4  print "Roosters", 100 - 25 * 3 % 4
5
6  print "Now I will count the eggs:"
7
8  print 3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6
9
10 print "Is it true that 3 + 2 < 5 - 7?"
11
12 print 3 + 2 < 5 - 7
13
14 print "What is 3 + 2?", 3 + 2
15 print "What is 5 - 7?", 5 - 7
16
17 print "Oh, that's why it's False."
18
19 print "How about some more."
20
21 print "Is it greater?", 5 > -2
22 print "Is it greater or equal?", 5 >= -2
23 print "Is it less or equal?", 5 <= -2
```

应该看到的结果

习题3 会话

\$ python ex3.py

I will now count my chickens:

Hens 30

Roosters 97

Now I will count the eggs:

7

Is it true that $3 + 2 < 5 - 7$?

False

What is $3 + 2$? 5

What is $5 - 7$? -2

Oh, that's why it's False.

How about some more.

Is it greater? True

Is it greater or equal? True

Is it less or equal? False

附加练习

- 1.每一行的上面使用#为自己写一个注释，说明一下这一行的作用。
- 2.记得习题0吧？用里边的方法运行Python，然后使用刚才学到的运算符号，把Python当做计算器玩玩。
- 3.自己找个想要计算的东西，写一个.py文件把它计算出来。
- 4.有没有发现计算结果是“错”的呢？计算结果只有整数，没有小数部分。研究一下这是为什么，搜索一下“浮点数”（floating point number）是什么东西。
- 5.使用浮点数重写一遍ex3.py，让它的计算结果更准确。（提示：20.0是一个浮点数。）

常见问题回答

为什么%是求余数符号，而不是百分号？

很大程度上只是因为设计人员选择了这个符号而已。正常写作时它是百分号没错，在编程中除法我们用了/，而求余数又恰恰选择了%这个符号，仅此而已。

%是怎么工作的？

换个说法就是“X除以Y还剩余J”，例如“100除以16还剩4”。%运算的结果就是J这部分。

运算优先级是怎样的？

在美国，我们用 PEMDAS 这个简称来辅助记忆，它的意思是“括号（Parentheses）、指数（Exponents）、乘（Multiplication）、除（Division）、加（Addition）、减（Subtraction）”，这也是Python里的运算优先级。

为什么/（除法）算出来的比实际小？

其实不是没算对，而是它将小数部分丢弃了，试试 $7.0 / 4.0$ 和 $7 / 4$ 比较一下，你就看出不同了。

习题4 变量和命名

你已经学会了`print`和算术运算。下一步要学的是“变量”（`variable`）。在编程中，变量只不过是用来指代某个东西的名字。程序员通过使用变量名可以让他们的程序读起来更像自然语言。而且因为程序员的记性都不怎么好，变量名可以让他们更容易记住程序的内容。如果他们没有在写程序时使用好的变量名，在下一次读到原来写的代码时他们会大为头疼的。

如果被这个习题难住了的话，想想之前教过的，要注意找到不同点、关注细节。

- 1.在每一行的上面写一行注释，给自己解释一下这一行的作用。
- 2.倒着读你的.py文件。
- 3.朗读你的.py文件，将每个字符也朗读出来。

ex4.py

```
1 cars = 100
2 space_in_a_car = 4.0
3 drivers = 30
4 passengers = 90
5 cars_not_driven = cars - drivers
6 cars_driven = drivers
7 carpool_capacity = cars_driven * space_in_a_car
8 average_passengers_per_car = passengers / cars_driven
9
```

```
10
11 print "There are", cars, "cars available."
12 print "There are only", drivers, "drivers available."
13 print "There will be", cars_not_driven, "empty cars today."
14 print "We can transport", carpool_capacity, "people today."
15 print "We have", passengers, "to carpool today."
16 print "We need to put about", average_passengers_per_car, "in
each car."
```

注意 `space_in_a_car` 中的 `_` 是下划线（underscore）字符。如果你不知道怎样输入这个字符的话就自己研究一下。这个字符在变量里通常被用作假想的空格，用来隔开单词。

应该看到的结果

习题4 会话

```
$ python ex4.py
```

There are 100 cars available.

There are only 30 drivers available.

There will be 70 empty cars today.

We can transport 120.0 people today.

We have 90 to carpool today.

We need to put about 3 in each car.

附加练习

当我刚开始写这个程序时我犯了个错误，Python告诉我这样的错误信息：

Traceback (most recent call last):

File "ex4.py", line 8, in <module>

```
average_passengers_per_car = car_pool_capacity /  
passenger
```

NameError: name 'car_pool_capacity' is not defined

用你自己的话解释一下这个错误信息，解释时记得使用行号，而且要说明原因。

下面是更多的附加练习。

1.我在程序里用了4.0作为 `space_in_a_car` 的值，这样做有必要吗？如果只用4会有什么问题？

2.记住4.0是一个“浮点数”，自己研究一下这是什么意思。

3.在每一个变量赋值的上一行加上一行注释。

4.记住=的名字是等于，它的作用是为东西取名。

5.记住_是下划线字符。

6.将Python作为计算器运行起来，就跟以前一样，不过这一次在计算过程中使用变量名来做计算，常见的变量名有i、x、j等。

常见问题回答

=（单等号）和==（双等号）有什么不同？

=的作用是将右边的值赋给左边的变量名。==的作用是检查左右两边是否相等。习题27中你会学到==的用法。

写成**x=100**而非**x = 100**也没关系吧？

是可以这样写，但这种写法不好。操作符两边加上空格会让代码更容易阅读。

词语间的空格有没有办法不让**print**打印出来？

你可以通过这样的方法实现：`print "Hey %s there." % "you"`，后面马上就会讲到。

怎样倒着读代码？

很简单，假如说你的代码有16行，你就从第16行开始，和我的第16行比对，接着比对第15行，依此类推，直到全部检查完。

为什么**space**用了**4.0**？

这个主要就是为了让你见识一下浮点数，并且提出这个问题。看看附加练习吧。

习题5 更多的变量和打印

我们现在要键入更多的变量并且把它们打印出来。这次我们将使用一个叫“格式化字符串”（`format string`）的东西。每一次你使用双引号（`"`）把一些文本引用起来，就创建了一个字符串。字符串是程序向人展示信息的方式。你可以打印（显示）它们，可以将它们写入文件，还可以将它们发送给网站服务器，很多事情都是通过字符串交流实现的。

字符串是非常好用的东西，所以在这个习题中你将学会如何创建包含变量内容的字符串。使用专门的格式和语法把变量的内容放到字符串里，相当于来告诉Python：“嘿，这是一个格式化字符串，把这些变量放到那几个位置。”

一样的，即使你读不懂这些内容，只要一字不差地键入就可以了。

ex5.py

```
1 my_name = 'Zed A.Shaw'
2 my_age = 35 # not a lie
3 my_height = 74 # inches
4 my_weight = 180 # lbs
5 my_eyes = 'Blue'
6 my_teeth = 'White'
7 my_hair = 'Brown'
8
9 print "Let's talk about %s." % my_name
10 print "He's %d inches tall." % my_height
```

```
11 print "He's %d pounds heavy." % my_weight
12 print "Actually that's not too heavy."
13 print "He's got %s eyes and %s hair." % (my_eyes, my_hair)
14 print "His teeth are usually %s depending on the coffee." %
my_teeth
15
16 # this line is tricky, try to get it exactly right
17 print "If I add %d, %d, and %d I get %d." % (
18     my_age, my_height, my_weight, my_age + my_height +
my_weight)
```

警告 如果你使用了非ASCII字符而且遇到了编码错误，记得在最顶端加上# -*- coding:utf-8 -*-。

应该看到的结果

习题5 会话

```
$ python ex5.py
```

Let's talk about Zed A.Shaw.

He's 74 inches tall.

He's 180 pounds heavy.

Actually that's not too heavy.

He's got Blue eyes and Brown hair.

His teeth are usually White depending on the coffee.

If I add 35, 74, and 180 I get 289.

附加练习

1.修改所有的变量名字，把它们前面的my_去掉。确认将每一个地方都改掉，不只是使用=赋值过的地方。

2.试着使用更多的格式化字符。例如，%r就是非常有用的一个，它的含义是：“不管什么都打印出来。”

3.在网上搜索所有的Python格式化字符。

4.试着使用变量将英寸和磅转换成厘米和千克。不要直接键入答案。使用Python的数学计算功能来完成。

常见问题回答

这样定义变量行不行：**`1 = 'Zed Shaw'`**？

不行。1不是一个有效的变量名称。变量名要以字母开头，所以a1可以，但1不行。

`%s`、**`%r`**和**`%d`**是做什么的？

后面你会学到更多，现在可以告诉你的是，它们是一种“格式控制工具”。它们告诉Python把右边的变量带到字符串中，并且把变量的值放到%s所在的位置上。

还是不懂，“格式控制工具”到底是什么？

教你学编程的一个问题就是，你需要先学会编程，才能读懂我的一些描述。我解决这个问题的方法是让你先去做一些事情，后面我再解释。当你碰到类似的问题时，把它们记录下来，看我是不是会在后面解释它们。

如何将浮点数四舍五入？

你可以使用round()函数，如round(1.7333)。

我遇到了错误**`TypeError: 'str' object is not callable`**。

很有可能你是漏写了字符串和变量之间的%。

为什么我还是不明白？

试着将脚本里的数字看成是你自己测量出来的数据，这样会很奇怪，但是多少会让你有身临其境的感觉，从而帮助你理解一些东西。

习题6 字符串和文本

虽然你已经在程序中写过字符串了，但是你还不了解它们的用处。在这个习题中我们将使用复杂的字符串来建立一系列变量，从中你将学到它们的用途。首先，我们解释一下字符串是什么。

字符串通常是指你想要展示给别人的或者是想要从程序里“导出”的一小段字符。**Python**可以通过文本里的双引号（"）或者单引号（'）识别出字符串来。这在以前的打印练习中你已经见过很多次了。如果你把单引号或者双引号括起来的文本放到 `print` 后面，它们就会被**Python**打印出来。

字符串可以包含之前已经见过的格式化字符。你只要将格式化的变量放到字符串中，紧跟着一个百分号%，再紧跟着变量名即可。唯一要注意的地方是，如果你想要在字符串中通过格式化字符放入多个变量，需要将变量放到圆括号（()）中，而且变量之间用逗号（,）隔开。就像你逛商店说“我要买牛奶、鸡蛋、面包、清汤”一样，只不过程序员的语法是“(milk, eggs, bread, soup)”。

我们将键入大量的字符串、变量、格式化字符，并且将它们打印出来。我们还将练习使用简写的变量名。程序员喜欢使用恼人的难读的简写来节约打字时间，所以我们就提早学会这个，这样你就能读懂并且写出这些东西了。

ex6.py

```
1 x = "There are %d types of people." % 10
2 binary = "binary"
```



```
3  do_not = "don't"
4  y = "Those who know %s and those who %s." % (binary, do_not)
5
6  print x
7  print y
8
9  print "I said: %r." % x
10 print "I also said: '%s'." % y
11
12 hilarious = False
13 joke_evaluation = "Isn't that joke so funny?! %r"
14
15 print joke_evaluation % hilarious
16
17 w = "This is the left side of..."
18 e = "a string with a right side."
19
20 print w + e
```

应该看到的结果

习题6 会话

```
$ python ex6.py
```

```
There are 10 types of people.
```

```
Those who know binary and those who don't.
```

```
I said: 'There are 10 types of people.'
```

```
I also said: 'Those who know binary and those who don't.'
```

```
Isn't that joke so funny?! False
```

```
This is the left side of...a string with a right side.
```

附加练习

- 1.通读这段程序，在每一行的上面写一行注释，给自己解释一下这一行的作用。
- 2.找到所有“把一个字符串放进另一个字符串”的位置。总共有4个地方。
- 3.你确定只有4个位置吗？你怎么知道的？没准儿我骗你呢。
- 4.解释一下为什么w和e用+连起来就可以生成一个更长的字符串。

常见问题回答

%r和**%s**有什么不同？

%r用来做调试（debug）比较好，因为它会显示变量的原始数据（raw data），而**%s**和其他的符号则是用来向用户显示输出的。

既然有**%r**了，为什么还要用**%s**和**%d**？

%r用来调试最好，而其他格式符则是用来向用户显示变量的。

如果你觉得很好笑，可不可以写一句**hilarious = True**？

可以。在习题27中你会学到关于布尔函数的更多知识。

为什么你在有些字符串上用单引号而在别的字符串上没有用？

很大程度上只是个风格问题，我的风格就是在双引号的字符串中使用单引号，比如代码的第10行就是这样做的。

我遇到了错误 **TypeError: not all arguments converted during string formatting**。

确定每一行代码都完全正确。发生这种错误是因为你的字符串里的**%**格式化字符数量比后面给的变量多，仔细检查一下哪里写错了。

习题7 更多打印

现在我们将做一批练习，在练习的过程中你需要键入代码，并且让它们运行起来。我不会解释太多，因为这个习题的内容都是以前熟悉过的。这个习题的目的是巩固你学到的东西。几个练习后再见。不要跳过这些练习。不要复制粘贴！

ex7.py

```
1  print "Mary had a little lamb."
2  print "Its fleece was white as %s." % 'snow'
3  print "And everywhere that Mary went."
4  print "." * 10    # what'd that do?
5
6  end1 = "C"
7  end2 = "h"
8  end3 = "e"
9  end4 = "e"
10 end5 = "s"
11 end6 = "e"
12 end7 = "B"
13 end8 = "u"
14 end9 = "r"
15 end10 = "g"
16 end11 = "e"
```

```
17 end12 = "r"
18
19 # watch that comma at the end. try removing it to see what
happens
20 print end1 + end2 + end3 + end4 + end5 + end6,
21 print end7 + end8 + end9 + end10 + end11 + end12
```

应该看到的结果

习题7 会话

```
$ python ex7.py
```

```
Mary had a little lamb.
```

```
Its fleece was white as snow.
```

```
And everywhere that Mary went.
```

```
...
```

```
Cheese Burger
```

附加练习

对于接下来几个习题，附加练习是一样的。

- 1.倒着阅读这段代码，在每一行的上面加一行注释。
- 2.倒着朗读出来，找出自己的错误。
- 3.从现在开始，把你的错误记录下来，写在一张纸上。
- 4.在开始下一个习题时，阅读一遍你记录下来的错误，并且尽量避免在下一个习题中再犯同样的错误。
- 5.记住，每个人都会犯错误。程序员和魔术师一样，他们希望大家认为他们从不犯错，不过这只是表象而已，他们每时每刻都在犯错。

常见问题回答

“**end**”语句是什么原理？

没有什么“end语句”，只是变量名里带了个“end”而已。

为什么要用一个叫'**snow**'的变量？

其实不是变量，而是一个内容为单词snow的字符串而已。变量名是不会带引号的。

你在附加练习**1**里说在每一行代码的上面写一条注释，一定要这样做吗？

不是。一般情况下加注释只是为了解释难懂的代码，或者注明为什么代码要这么写。一般来说后者更为重要。遇到代码的确每一行都很难懂的特殊情况，加注释是正确的选择。在这里，我主要是为了让你逐渐学会将代码翻译成日常语言。

创建字符串时是不是单引号和双引号都可以，它们有什么不同用途吗？

Python里边两种都是可以的，不过一般单引号会被用来创建简短的字符串，如'a'、'snow'这样的。

可以不用逗号，将最后两行写成一行输出吗？

当然可以，不过这样一来这行的长度就超过80个字符了，从Python代码风格方面来讲这样做是不好的。

习题8 打印，打印

ex8.py

```
1  formatter = "%r %r %r %r"
2
3  print formatter % (1, 2, 3, 4)
4  print formatter % ("one", "two", "three", "four")
5  print formatter % (True, False, False, True)
6  print formatter % (formatter, formatter, formatter, formatter)
7  print formatter % (
8      "I had this thing.",
9      "That you could type up right.",
10     "But it didn't sing.",
11     "So I said goodnight."
12 )
```

应该看到的结果

习题8 会话

```
$ python ex8.py
```

```
1 2 3 4
```

```
'one' 'two' 'three' 'four'
```

```
True False False True
```

```
'%r %r %r %r' '%r %r %r %r' '%r %r %r %r' '%r %r %r %r'
```

```
'I had this thing.' 'That you could type up right.' "But it didn't sing."
```

```
'So I said goodnight.'
```

附加练习

- 1.自己检查结果，记录你犯的错误，并且在下一个习题中尽量不犯同样的错误。
- 2.注意，最后一行输出既有单引号又有双引号，你觉得它是如何工作的？

常见问题回答

我应该用**%s**还是用**%r**？

你应该用**%s**，只有在想要获取某些东西的调试信息时才能用到**%r**。**%r**给你的是变量的“程序员原始版本”，又被称作“原始表示”（raw representation）。

为什么“**one**”要用引号，而**True**和**False**不需要？

因为**True**和**False**是Python的关键字，用来表示真和假的概念。如果加了引号，它们就变成了字符串，也就无法实现它们本来的功能了。习题27中会有详细说明。

我在字符串中包含了中文（或者其他非**ASCII**字符），可是**%r**打印出的是乱码？

使用**%s**打印就行了。

为什么**%r**有时打印出来的是单引号，而我实际用的是双引号？

Python会用最有效的方式打印出字符串，而不是完全按照你写的方式来打印。这样做对于**%r**来说是可以接受的，因为它是用于调试和排错的，没必要非打印出多好看的格式。

为什么**Python 3**里这些都不灵？

别使用Python 3。使用Python 2.7或更新的版本，尽管Python 2.6应该也没问题。

可不可以使用**IDLE**运行这段代码？

不行。你应该学习使用命令行。命令行对学习编程很重要，而且是学习编程的绝佳初始环境。**IDLE**在本书后面会让你失望的。

习题9 打印，打印，打印

ex9.py

```
1  # Here's some new strange stuff, remember type it exactly.
2
3  days = "Mon Tue Wed Thu Fri Sat Sun"
4  months = "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"
5
6  print "Here are the days: ", days
7  print "Here are the months: ", months
8
9  print """
10  There's something going on here.
11  With the three double-quotes.
12  We'll be able to type as much as we like.
13  Even 4 lines if we want, or 5, or 6.
14  """
```

应该看到的结果

习题9 会话

```
$ python ex9.py
```

```
Here are the days: Mon Tue Wed Thu Fri Sat Sun
```

```
Here are the months: Jan
```

```
Feb
```

```
Mar
```

```
Apr
```

```
May
```

```
Jun
```

```
Jul
```

```
Aug
```

```
There's something going on here.
```

```
With the three double-quotes.
```

```
We'll be able to type as much as we like.
```

```
Even 4 lines if we want, or 5, or 6.
```

附加练习

自己检查结果，记录你犯的错误，并且在下一个练习中尽量不犯同样的错误。

常见问题回答

怎样将每个月份显示在新的一行？

字符串以\n开始就可以了，像这

样：`"\nJan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"`。

为什么使用%r时\n换行就不灵了？

%r就是这个样子，它打印出的是你写出来的方式（或近似方式）。

它是用来调试的“原始”格式。

为什么在三引号之间加入空格就会出错？

你必须写成`"""`而不是`" "`，引号之间不能有空格。

我的大部分错误都是拼写错误，是不是我太笨了？

对于初学者甚至进阶学员来说，大部分编程中的错误都是拼写错误、录入错误或者没把别的一些简单东西弄对。

习题10 那是什么

在习题9中我带你接触了一些新东西。我让你看到两种让字符串扩展到多行的方法。第一种方法是在月份之间用\n隔开。这两个字符的作用是在该位置上放入一个“换行”（new line）字符。

使用反斜杠（\）可以将难打印出来的字符放到字符串。针对不同的符号有很多这样的所谓“转义序列”（escape sequences），但有一个特殊的转义序列，就是双反斜杠（\\）。这两个字符组合会打印出一个反斜杠来。接下来我们试几个这样的转义序列，你就知道这些转义序列的意义了。

另外一种重要的转义序列是用来将单引号（'）和双引号（"）转义。想象你有一个用双引号括起来的字符串，你想要在字符串的内容里再添加一组双引号进去，比如，你想说"I understand joe."，Python 就会认为"understand"前后的两个引号是字符串的边界，从而把字符串弄错。你需要一种方法告诉Python，字符串里边的双引号不是真正的双引号。

要解决这个问题，需要将双引号和单引号转义，让Python将引号也包含到字符串里边去。下面是一个例子：

```
"I am 6'2\" tall." # 将字符串中的双引号转义
```

```
'I am 6'2" tall.' # 将字符串中的单引号转义
```

第二种方法是使用“三引号”，也就是 `"""`，你可以在一组三引号之间放入任意多行的文字。接下来你将看到用法。

```
1 tabby_cat = "\tI'm tabbed in."
2 persian_cat = "I'm split\non a line."
3 backslash_cat = "I'm \\ a \\ cat."
4
5 fat_cat = """
6 I'll do a list:
7 \t* Cat food
8 \t* Fishies
9 \t* Catnip\n\t* Grass
10 """
11
12 print tabby_cat
13 print persian_cat
14 print backslash_cat
15 print fat_cat
```

应该看到的结果

注意你打印出来的制表符（tab），这个习题中的文字间隔对于答案的正确性是很重要的。

习题10 会话

```
$ python ex10.py
```

```
    I'm tabbed in.
```

```
I'm split
```

```
on a line.
```

```
I'm \ a \ cat.
```

```
I'll do a list:
```

```
    * Cat food
```

```
    * Fishies
```

```
    * Catnip
```

```
    * Grass
```

转义序列

下面列出了Python支持的所有转义序列。很多你也许不会用到，不过还是要记住它们的格式和功能。试着在字符串中应用它们，看看你是否让它们起作用。

转义字符	功能
\\	反斜杠 (\)
\'	单引号 (')
\"	双引号 (")
\a	ASCII响铃符 (BEL)
\b	ASCII退格符 (BS)
\f	ASCII进纸符 (FF)
\n	ASCII换行符 (LF)
\N{name}	Unicode数据库中的字符名，其中name是它的名字，仅适用Unicode
\r	ASCII回车符 (CR)
\t	ASCII水平制表符 (TAB)
\uxxxx	值为16位十六进制值xxxx的字符（仅适用Unicode）
\Uxxxxxxxx	值为32位十六进制值xxxxxxxx的字符（仅适用Unicode）
\v	ASCII垂直制表符 (VT)
\ooo	值为八进制值ooo的字符
\xhh	值为十六进制数hh的字符

试着运行下面一小段代码看看结果：

```
while True:
```

```
    for i in ["/","-","|","\\","|"]:
```

```
        print "%s\r" % i,
```

附加练习

- 1.把这些转义字符记录到卡片上，并记住它们的含义。
- 2.使用'''（三个单引号）取代三个双引号，你能想出什么场合下应该用它而不是用""吗？
- 3.将转义序列和格式化字符串组合到一起，创建一种更复杂的格式。
- 4.记得%r 格式化字符串吗？使用%r 搭配单引号和双引号转义字符打印一些字符串出来。将%r和%s比较一下。注意到了吗？%r打印出来的是你作为程序员写在脚本里的东西，而%s打印的是你作为用户应该看到的東西。

常见问题回答

我还没完全搞明白上一习题，我可以继续吗？

可以，继续向下看，看完一部分后回头看自己以前在笔记本上记下来的不懂的知识点，看是不是已经明白了。有时你可能还需要回到前面的习题中重新复习一遍。

`\\`和别的符号相比有什么特别之处吗？

并无特别，这样只是为了输出一个反斜杠（`\`），想想为什么要把它写成两杠。

`//`和`\n`怎么不灵？

因为你用了斜杠（`/`）而不是反斜杠（`\`），它们是不一样的字符，功能也完全不同。

使用了`%r`后转义序列都不灵了。

因为`%r`打印出的是你写到代码里的原始字符串，其中会包含原始的转义字符。你应该使用`%s`，记住这条：`%r`用于调试，`%s`用于显示。

附加练习3说是要组合什么的，是什么意思？

我想让你明白的一点是，所有这些习题中教你的东西都可以组合起来帮你解决问题。把你学过的格式化字符串的知识和你新学到的转义字符的知识组合起来，写一些代码。

`'`和`''''`哪个好？

风格问题。现在你就用`'`吧，以后遇到再说。有时候用某一种可能会更美观，有时候你要遵循之前的写法从而让整个项目代码风格一致，看具体情况。

习题11 提问

我已经出过很多打印相关的习题，让你习惯写简单的东西，但简单的东西都有点儿无聊，现在该跟上脚步了。我们现在要做的是把数据读到你的程序里去。这可能对你有点儿难度，你可能一下子不明白，不过你要相信我，无论如何把习题做了再说。只要做几个习题你就明白了。

一般软件做的事情主要就是下面几条。

- 1.接收人的输入。
- 2.改变输入的内容。
- 3.打印出改变了的内容。

到目前为止你只做了打印，但还不会接收或者修改人的输入。你也许还不知道“输入”是什么意思。所以闲话少说，我们还是开始做点儿练习看你能不能明白。下一个习题里边我们会给你更多的解释。

ex11.py

```
1  print "How old are you?",
2  age = raw_input()
3  print "How tall are you?",
4  height = raw_input()
5  print "How much do you weigh?",
6  weight = raw_input()
7
8  print "So, you're %r old, %r tall and %r heavy." % (
9      age, height, weight)
```

注意 我在每行`print`后面加了个逗号（,），这样的话`print`就不会输出换行符而结束这一行跑到下一行去了。

应该看到的结果

习题11 会话

```
$ python ex11.py
```

```
How old are you? 38
```

```
How tall are you? 6'2"
```

```
How much do you weigh? 180lbs
```

```
So, you're '38' old, '6'2'" tall and '180lbs' heavy.
```

附加练习

1. 上网查一下Python的raw_input实现的是什么功能。
2. 你能找到它的别的用法吗？测试一下你上网搜到的例子。
3. 用类似的格式再写一段代码，把问题改成你自己的问题。
4. 和转义序列有关的，想想为什么最后一行'6'2'"里边有一个\'序列。单引号需要被转义，从而防止它被识别为字符串的结尾。有没有注意到这一点？

常见问题回答

如何读取用户输入的数进行计算？

这个有点算高级话题了。试试 `x = int(raw_input())`，它会把用户输入的字符串用`int()`转换成整数。

我把身高写到原始输入`raw_input("6'2")`怎么不灵？

不应该写成这样，只有从命令行输入才可以。首先回去把代码写成和我的一模一样，然后运行脚本，当脚本暂停下来的时候，用键盘输入你的身高。这样做就可以了。

为什么第**8**行要新写一行而不是放在一整行里边？

这样是为了保持每行不多于 80 个字符，Python 程序员喜欢这样的风格。放在一整行里也不是不行。

`input()`和**`raw_input()`**有何不同？

`input()`函数会把你输入的东西当做 Python 代码进行处理，这么做会有安全问题，你应该避开这个函数。

打印出来后我的字符串前面有个**`u`**，像**`u'35'`**这样。

它表示Python告诉你你的字符串是Unicode。使用**`%s`**就一切正常了。

习题12 提示别人

当你键入 `raw_input()` 的时候，你需要输入一个括号。这和你格式化输出两个以上变量时的情况有点类似，比如说 `"%s %s" % (x, y)` 里边就有括号。对于 `raw_input` 而言，你还可以让它显示出一个提示，从而告诉别人应该输入什么东西。你可以在 `()` 之间放入一个你想要作为提示的字符串，如下所示：

```
y = raw_input("Name? ")
```

这句话会用“Name?”提示用户，然后将用户输入的结果赋值给变量 `y`。这就是我们提问用户并且得到答案的方式。

也就是说，我们的上一个练习可以使用 `raw_input` 重写一次。所有的提示都可以通过 `raw_input` 实现。

ex12.py

```
1 age = raw_input("How old are you? ")
2 height = raw_input("How tall are you? ")
3 weight = raw_input("How much do you weigh? ")
4
5 print "So, you're %r old, %r tall and %r heavy." % (
6     age, height, weight)
```

应该看到的结果

习题12 会话

```
$ python ex12.py
```

```
How old are you? 38
```

```
How tall are you? 6'2"
```

```
How much do you weight? 180lbs
```

```
So, you're '38' old, '6'2'" tall and '180lbs' heavy.
```

附加练习

1.在终端上运行你的程序，然后在终端上输入 `pydoc raw_input` 看它说了些什么。

如果你用的是Windows，那就试一下`python -m pydoc raw_input`。

2.键入q退出pydoc。

3.上网找一下pydoc命令是用来做什么的。

4.使用pydoc再看一下`open`、`file`、`os`和`sys`的含义。看不懂没关系，只要通读一下，记下你觉得有趣的知识点就行了。

常见问题回答

运行**pydoc**时我怎么遇到**SyntaxError: invalid syntax**?

你没有从命令行运行**pydoc**，很可能是从python里运行的。退出python试试。

我的**pydoc**为什么不像你的那样会暂停?

有时文档很短，一屏就显示完了，这时**pydoc**就不会暂停。

运行**pydoc**看到**more is not recognized**。

Windows的有些版本中没有这个命令，也就是说你没法用**pydoc**了。跳过这些附加练习，上网去搜索Python文档吧。

%r和**%s**该用哪个?

记住**%r**是调试专用，它显示的是“原始表示”出来的字符，而**%s**是为了给用户显示。这个问题以后我就不再回答了，你要牢牢记住。这个问题是人们重复问的最多的，如果同一个问题要问很多遍，那说明你没记住你该记住的东西。别问了，现在要你必须记住。

写成**print "How old are you?" , raw_input()**为什么不行?

你觉得可以，但Python不这么认为。我唯一能给你的答案是：不行就是不行。

习题13 参数、解包和变量

这个习题中，我们将讲到另外一种将变量传递给脚本的方法（所谓脚本，就是你编写的.py程序）。你已经知道，如果要运行 `ex13.py`，只要在命令行运行 `python ex13.py` 就可以了。这条命令中的`ex13.py`部分就是所谓的“参数”（argument），我们现在要做的就是写一个可以接收参数的脚本。

输入下面的程序，后面我会详细解释。

`ex13.py`

```
1  from sys import argv
2
3  script, first, second, third = argv
4
5  print "The script is called:", script
6  print "Your first variable is:", first
7  print "Your second variable is:", second
8  print "Your third variable is:", third
```

在第1行我们有一个import语句，这是你将Python的特性引入脚本的方法。Python不会一下子将它所有的特性给你，而是让你需要什么就调用什么。这样不但可以让你的程序保持很小，而且以后其他程序员读你的代码时，这些import可以作为文档查阅。

Argv即所谓的“参数变量”（argument variable），这是一个非常标准的编程术语。在其他编程语言中也可以看到它。这个变量保存着你运行

Python 脚本时传递给 Python 脚本的参数。通过后面的练习，你将对它有更多的了解。

第 3 行将 `argv`“解包”（`unpack`），与其将所有参数放到同一个变量下面，不如将每个参数赋值给一个变量：`script`、`first`、`second`和`third`。这也许看上去有些奇怪，不过“解包”可能是最好的描述方式了。它的含义很简单：“把`argv`中的东西解包，将所有的参数依次赋值给左边的这些变量。”

接下来就是正常的打印了。

等一下！“特性”还有另外一个名字

前面我们使用import让你的Python程序实现更多的特性，虽然我们称其为“特性”，但实际上没人把它称为“特性”。我希望你可以在没接触到正式术语的时候就弄懂它的功能。在继续学习之前，你需要知道它们的真正名称：模块（module）。

从现在开始我们将把这些导入（import）的特性称为模块。你将看到类似这样的说法：“你需要把sys模块导入进来。”也有人将它们称作“库”（library），不过我们还是叫它们模块吧。

应该看到的结果

用下面的方法运行你的程序（注意必须传递3个命令行参数）。

习题13 会话

```
$ python ex13.py first 2nd 3rd
```

```
The script is called: ex13.py
```

```
Your first variable is: first
```

```
Your second variable is: 2nd
```

```
Your third variable is: 3rd
```

如果你每次使用不同的参数运行，你将看到下面的结果。

习题13 会话

```
$ python ex13.py stuff things that
```

```
The script is called: ex13.py
```

```
Your first variable is: stuff
```

```
Your second variable is: things
```

```
Your third variable is: that
```

```
$
```

```
$ python ex13.py apple orange grapefruit
```

```
The script is called: ex13.py
```

```
Your first variable is: apple
```

```
Your second variable is: orange
```

```
Your third variable is: grapefruit
```

其实可以将first、second、third替换成任意三样东西。你可以将它们换成任意你想要的东西。

如果没有运行对，你将看到如下错误。

习题13 会话

```
$ python ex13.py first 2nd
```

```
Traceback (most recent call last):
```

```
  File "ex13.py", line 3, in <module>
```

```
    script, first, second, third = argv
```

```
ValueError: need more than 3 values to unpack
```

如果运行脚本时提供的参数的个数不对，你就会看到上述错误信息（这次我只用了first 2nd）。“need more than 3 values to unpack”这个错误信息告诉你参数数量不足。

附加练习

- 1.给你的脚本三个以下的参数。看看会得到什么错误信息。试着解释一下。
- 2.再写两个脚本，其中一个接收更少的参数，另一个接收更多的参数，在参数解包时给它们取一些有意义的变量名。
- 3.将raw_input和argv一起使用，让你的脚本从用户那里得到更多的输入。
- 4.记住，“模块”为你提供额外的特性。多读几遍把“模块”这个词记住，因为后面还会用到它。

常见问题回答

运行时我遇到了**ValueError: need more than 1 value to unpack**。

记住，有一个很重要的技能是注重细节。如果你仔细阅读并且完整重复了“应该看到的结果”部分的命令参数，你就不会看到这样的错误信息了。

argv和**raw_input()**有什么不同？

不同点在于用户输入的时机。如果参数是在用户执行命令时就要输入，那就是 **argv**，如果是在脚本运行过程中需要用户输入，那就使用 **raw_input()**。

命令行参数是字符串吗？

是的，就算你在命令行输入数字，你也需要用**int()**把它先转成数字，和在**raw_input()**里一样。

命令行该怎么使用？

这个你应该已经学会了才对。如果你还没学会，就去读读附录的“命令行快速入门”吧。

argv和**raw_input()**怎么不能合起来用。

别想太多了。在脚本结尾加两行**raw_input()**随便读取点用户输入然后打印出来就行了，然后再慢慢在同一脚本中用各种方法玩玩这两样东西。

为什么**raw_input('? ') = x**不灵？

因为你写反了。照着我的写就没问题了。

习题14 提示和传递

让我们使用 `argv` 和 `raw_input` 一起来向用户提一些特别的问题。下一个习题你会学习如何读写文件，这个习题是下一个习题的基础。在这个习题里我们将用略微不同的方法使用 `raw_input`，让它显示一个简单的>作为提示符。这和一些游戏中的方式类似，如 `Zork`和`Adventure`这两款游戏。

ex14.py

```
1  from sys import argv
2
3  script, user_name = argv
4  prompt = '> '
5
6  print "Hi %, I'm the %s script." % (user_name, script)
7  print "I'd like to ask you a few questions."
8  print "Do you like me %s?" % user_name
9  likes = raw_input(prompt)
10
11 print "Where do you live %s?" % user_name
12 lives = raw_input(prompt)
13
14 print "What kind of computer do you have?"
15 computer = raw_input(prompt)
```

```
16
17 print ""
18 Alright, so you said %r about liking me.
19 You live in %r. Not sure where that is.
20 And you have a %r computer. Nice.
21 "" % (likes, lives, computer)
```

我们将用户提示符设置为变量prompt，这样就不需要在每次用到raw_input时重复输入提示用户的字符了。而且，如果你要将提示符修改成别的字符串，只要改一个位置就可以了。

非常顺手吧。

应该看到的结果

当你运行这个脚本时，记住需要把你的名字赋给这个脚本，让argv参数接收到你的名字。

习题14 会话

```
$ python ex14.py Zed
Hi Zed, I'm the ex14.py script.
I'd like to ask you a few questions.
Do you like me Zed?
> Yes
Where do you live Zed?
> San Francisco
What kind of computer do you have?
> Tandy 1000
Alright, so you said 'Yes' about liking me.
You live in 'San Francisco'.Not sure where that is.
And you have a 'Tandy 1000' computer.Nice.
```

附加练习

- 1.查一下Zork和Adventure是两款什么样的游戏。看看能不能下载到一版，然后玩玩看。
- 2.将prompt变量改成完全不同的内容再运行一遍。
- 3.给你的脚本再添加一个参数，并使用这个参数。
- 4.确认你弄懂了三个引号"""可以定义多行字符串，而%是字符串的格式化工具。

常见问题回答

运行这段脚本时出现**SyntaxError: invalid syntax**。

再次说明，你应该在命令行上而不是在Python环境中运行脚本。如果你先键入了python然后试图键入python ex14.py Zed，就会出现这个错误，你这是在Python里运行Python。关掉窗口，重新键入python ex14.py Zed。

修改提示符是什么意思？

看变量定义prompt = '> '，将它改成一个不同的值。这个应该难不倒你，只是修改一个字符串而已，前面的13个习题都是围绕字符串的，自己花时间搞定。

发生错误**ValueError: need more than 1 value to unpack**。

记得上次我说过，你应该到“应该看到的结果”部分重复我的动作。把精力集中到我的输入，以及为什么我提供了一个命令行参数。

我可以用双引号定义prompt变量的值吗？

当然可以，试试看就知道了。

你有台Tandy计算机？

我小时候有过。

运行时这段脚本时出现**NameError: name 'prompt' is not defined**。

要么拼错了 prompt，要么漏写了这一行。回去比较你写的和我写的，从最后一行开始直至第一行。

怎样从IDLE中运行？

不要使用IDLE。

习题15 读取文件

你已经学过了`raw_input`和`argv`，这些是开始学习读取文件的必备基础。你可能需要多多实践才能明白它的工作原理，所以你要细心做练习，并且仔细检查结果。处理文件需要非常仔细，如果不仔细的话，可能会把有用的文件弄坏或者清空，导致前功尽弃。

这个习题涉及写两个文件：一个正常的`ex15.py`文件，另外一个是一个是`ex15_sample.txt`。第二个文件并不是脚本，而是供你的脚本读取的文本文件。下面是该文本文件的内容。

This is stuff I typed into a file.

It is really cool stuff.

Lots and lots of fun to have in here.

我们要做的是用我们的脚本“打开”该文件，然后打印出来。然而把文件名 `ex15_sample.txt`“写死”（`hardcode`）在代码中不是一个好主意，这些信息应该是用户输入的才对。如果我们遇到其他文件要处理，写死的文件名就会给你带来麻烦。我们的解决方案是，使用`argv`和`raw_input`从用户那里获取信息，从而知道要处理哪些文件。

`ex15.py`

```
1  from sys import argv
2
3  script, filename = argv
4
5  txt = open(filename)
```

```
6
7  print "Here's your file %r:" % filename
8  print txt.read()
9
10 print "Type the filename again:"
11 file_again = raw_input("> ")
12
13 txt_again = open(file_again)
14
15 print txt_again.read()
```

这个脚本中有一些新奇的玩意儿，我们来快速地过一遍。

代码的第 1~3 行使用 `argv` 来获取文件名，这个你应该已经很熟悉了。在接下来的第 5 行我们看到 `open` 这个新命令。现在请在命令行运行 `pydoc open` 来读读它的说明。你可以看到它和你自己的脚本或者 `raw_input` 命令类似，它会接收一个参数，并且返回一个值，你可以将这个值赋给一个变量。这就是你打开文件的过程。

第 7 行我们打印了一小行，但在第 8 行我们看到了新奇的东西。我们在 `txt` 上调用了一个函数。你从 `open` 获得的东西是一个 `file`（文件），文件本身也支持一些命令。它接收命令的方式是使用句点（`.`），紧跟着你的命令，然后是类似 `open` 和 `raw_input` 的参数。不同点是：当你说 `txt.read` 时，你的意思其实是：“嘿 `txt`！执行你的 `read` 命令，无需任何参数！”

脚本剩下的部分基本差不多，我就把剩下的分析作为附加练习留给你吧。

应该看到的结果

我创建了一个名为ex15_sample.txt的文件，并运行我的脚本。

习题15 会话

```
$ python ex15.py ex15_sample.txt
```

```
Here's your file 'ex15_sample.txt':
```

```
This is stuff I typed into a file.
```

```
It is really cool stuff.
```

```
Lots and lots of fun to have in here.
```

```
Type the filename again:
```

```
> ex15_sample.txt
```

```
This is stuff I typed into a file.
```

```
It is really cool stuff.
```

```
Lots and lots of fun to have in here.
```


附加练习

这个习题跨越有点大，所以要尽量做好这个习题的附加练习，然后再继续学习后面的内容。

1.在每一行的上面用注释说明这一行的用途。

2.如果你不确定答案，就问别人，或者上网搜索。大部分时候，只要搜索“python”加上你要搜的东西就能得到你要的答案。比如，搜索一下“python open”。

3.这里我使用了“命令”这个词，不过实际上它们也叫“函数”（function）和“方法”（method）。上网搜一下，看看其他人是怎么定义它们的。看不明白也没关系，迷失在别的程序员的知识海洋里是很正常的一件事情。

4.删掉第10~15行用到raw_input的部分，再运行一遍脚本。

5.只是用raw_input写这个脚本，想想哪种获取文件名称的方法更好，为什么。

6.运行pydoc file，向下滚动直到看见read()命令（函数/方法）。看到很多别的命令了吧，你可以找几条试试看。不需要看那些包含__（两个下划线）的命令，这些只是垃圾而已。

7.再次运行python，在提示符下使用open打开一个文件，这种open和read的方法也值得一学。

8.让你的脚本针对txt和txt_again变量执行一下close()。处理完文件后需要将其关闭，这是很重要的一点。

常见问题回答

txt = open(filename)返回的是文件的内容吗？

不是，它返回的是一个叫做“file object”的东西，你可以把它想象成20世纪50年代的大型计算机上可以见到的古老的磁带机或者现代的DVD 机。你可以随意访问内容的任意位置，然后读取这些内容，不过这个文件本身并不是它的内容。

我没法像你在附加练习7中说的那样在我的**Terminal/PowerShell**命令行下输入**Python**代码。

首先，在命令行输入python然后按回车键。现在你就在python环境中了。接下来你就可以输入并运行一句一句的代码。试着玩玩，如果想退出就输入quit()再敲回车。

from sys import argv是什么意思？

现在能告诉你的是，sys是一个软件包，这句话的意思是从该软件包中取出argv这个特性来，供我使用。后面你会学到更多相关的知识。

我把文件名写进去，写成**script, ex15_sample.txt = argv**，不过这样不灵。

这么做是错的。把代码写成和我一模一样的，然后照着我的方式从命令行运行。你不需要把文件名放到代码中，而是让Python把文件名当做参数接纳进去。

为什么文件打开了两次没有报错？

Python 不会限制你打开文件的次数，事实上，有时候多次打开同一个文件是一件必须的事情。

习题16 读写文件

如果你做了上一个习题的附加练习，应该已经了解了各种与文件相关的命令（方法/函数）。下面这些似乎我想让你记住的命令。

`close`——关闭文件。跟你编辑器的“文件”→“保存”是一个意思。

`read`——读取文件内容。你可以把结果赋给一个变量。

`readline`——读取文本文件中的一行。

`truncate`——清空文件，请小心使用该命令。

`write(stuff)`——将stuff写入文件。

这是你现在应该知道的重要命令。有些命令需要接收参数，这对我们并不重要。你只要记住`write`的用法就可以了。`write`需要接收一个字符串作为参数，从而将该字符串写入文件。

让我们来使用这些命令做一个简单的文本编辑器吧。

ex16.py

```
1  from sys import argv
2
3  script, filename = argv
4
5  print "We're going to erase %r." % filename
6  print "If you don't want that, hit CTRL-C (^C)."
7  print "If you do want that, hit RETURN."
8
9  raw_input("?")
```

```
10
11 print "Opening the file..."
12 target = open(filename, 'w')
13
14 print "Truncating the file.  Goodbye!"
15 target.truncate()
16
17 print "Now I'm going to ask you for three lines."
18
19 line1 = raw_input("line 1: ")
20 line2 = raw_input("line 2: ")
21 line3 = raw_input("line 3: ")
22
23 print "I'm going to write these to the file."
24
25 target.write(line1)
26 target.write("\n")
27 target.write(line2)
28 target.write("\n")
29 target.write(line3)
30 target.write("\n")
31
32 print "And finally, we close it."
33 target.close()
```

这个文件是够大的，大概是你键入过的最大的文件。所以慢慢来，仔细检查，让它能运行起来。有一个小技巧就是，你可以让你的脚本一部分一部分地运行起来。先写第1~8行，让它运行起来，再多运行5

行，再接着多运行几行，依此类推，直到整个脚本运行起来为止。

应该看到的结果

你将看到两样东西，第一样是你的新脚本的输出，具体如下。

习题16 会话

```
$ python ex16.py test.txt
```

```
We're going to erase 'test.txt'.
```

```
If you don't want that, hit CTRL-C (^C).
```

```
If you do want that, hit RETURN.
```

```
?
```

```
Opening the file...
```

```
Truncating the file.Goodbye!
```

```
Now I'm going to ask you for three lines.
```

```
line 1: Mary had a little lamb
```

```
line 2: It's fleece was white as snow
```

```
line 3: It was also tasty
```

```
I'm going to write these to the file.
```

```
And finally, we close it.
```

接下来打开你新建的文件（我的是test.txt）检查一下里边内容，怎么样，不错吧？

附加练习

- 1.如果你觉得自己没有弄懂，用我们的老办法，在每一行之前加上注释，为自己理清思路。就算不能理清思路，你也可以知道自己究竟哪里没弄明白。
- 2.写一段遇上上一个习题类似的脚本，使用`read`和`argv`读取你刚才新建的文件。
- 3.文件中重复的地方太多了。试着用一个`target.write()`将`line1`、`line2`、`line3`打印出来，你可以使用字符串、格式化字符及转义字符。
- 4.找出为什么我们需要给`open`多赋予一个'`w`'参数。提示：`open`对于文件的写入操作态度是安全第一，所以只有特别指定以后，它才会进行写入操作。
- 5.如果你用'`w`'模式打开文件，那么你是不是还要`target.truncate()`呢？阅读一下Python的`open`函数的文档找找答案。

常见问题回答

如果用了'**w**'参数，**truncate()**是必需的吗？

看看附加练习5。

'**w**'是什么意思？

它只是一个只有一个字符的特殊字符串，用来表示文件的访问模式。如果你用了'**w**'，那么你的文件就是“写入”（**write**）模式。除了'**w**'以外，我们还有'**r**'表示“读取”（**read**），'**a**'表示“追加”（**append**）。

还有哪些修饰符可以用来控制文件访问？

当前最重要的一个是+修饰符，你可以用它来实现 '**w+**'、'**r+**'和'**a+**'。这样可以把文件用同时读写的方法打开，每个符号会以不一样的方式实现文件内部的定位。

如果只写**open(filename)**，那就使用'**r**'模式打开吗？

是的，这是**open()**函数的默认模式。

习题17 更多文件操作

现在再学习几种文件操作。我们将编写一个Python脚本，将一个文件中的内容复制到另外一个文件中。这个脚本很短，不过它会让你对文件操作有更多的了解。

ex17.py

```
1  from sys import argv
2  from os.path import exists
3
4  script, from_file, to_file = argv
5
6  print "Copying from %s to %s" % (from_file, to_file)
7
8  # we could do these two on one line too, how?
9  in_file = open(from_file)
10  indata = in_file.read()
11
12  print "The input file is %d bytes long" % len(indata)
13
14  print "Does the output file exist? %r" % exists(to_file)
15  print "Ready, hit RETURN to continue, CTRL-C to abort."
16  raw_input()
17
```

```
18 out_file = open(to_file, 'w')
19 out_file.write(indata)
20
21 print "Alright, all done."
22
23 out_file.close()
24 in_file.close()
```

你应该很快注意到了，我们导入了又一个很好用的命令 `exists`。这个命令将文件名字符串作为参数，如果文件存在的话，它将返回 `True`；否则将返回 `False`。在本书的下半部分，我们将使用这个函数做很多的事情，不过现在你应该学会怎样通过`import`调用它。

通过使用 `import`，你可以在自己代码中直接使用其他更厉害的（通常是这样，不过也不尽然）程序员写的大量免费代码，这样你就不需要重写一遍了。

应该看到的结果

和你前面写的脚本一样，运行该脚本需要两个参数：一个是待复制的文件，另一个是要复制到的文件。如果使用以前的test.txt测试文件，我们将看到如下的结果。

习题17 会话

```
$ cat test.txt
This is a test file.
$
$ python ex17.py test.txt mew-file.txt
Copying from test.txt to mew-file.txt
The input file is 81 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort.
Alright, all done.
```

该命令对于任何文件都应该是有效的。试试操作一些别的文件，看看结果。不过，小心别把你的重要文件给弄坏了。

警告 我用了cat这个命令来显示文件的内容，你看到了吗？你可以从附录中学到如何做到这一点。

附加练习

1.再多读读和import相关的材料，启动python，试试这一条命令。试着看看自己能不能摸出点儿门道，当然了，即使弄不明白也没关系。

2.这个脚本实在是有点儿烦人。没必要在复制之前问你，也没必要在屏幕上输出那么多东西。试着删掉脚本的一些特性，让它使用起来更加友好。

3.看看你能把这个脚本改多短，我可以把它变成一行。

4.我使用了一个叫 `cat` 的东西，这个古老的命令的用处是将两个文件“拼接”（`concatenate`）到一起，不过实际上它最大的用途是打印文件内容到屏幕上。你可以通过`man cat`命令了解到更多信息。

5.使用Windows的你可以给自己找一个cat的替代品。关于man的东西就别想太多了，Windows下没有类似的命令。

6.找出为什么你需要在代码中写`output.close()`。

常见问题回答

为什么'**w**'要放在括号中？

因为这是一个字符串，你已经学过一阵子字符串了，确定自己真的学会了。

不可能把这写在一行里！

取决于你的行是怎么定义的，例如，这样That ; depends ; on ; how ; you ; define ; one ; line ; of ; code。

len()函数的功能是什么？

它会以数的形式返回你传递的字符串的长度。试试吧。

在我试图把代码写短时，我在最后关闭该文件时出现一个错误。

很可能是你写了 `indata = open(from_file).read()`，这意味着你无需再运行`in_file.close()`了，因为`read()`一旦运行，文件就会被Python关掉。

我觉得这个习题很难，这个是正常现象吗？

是的，再正常不过了。也许在你看到习题 36 之前，甚至读完整本书，编程对你来说都还是一件很难理解的事情。每个人的情况都不一样，坚持读书做练习，有问题的地方多研究，总会弄明白的。慢工出细活。

我遇到了**Syntax:EOL while scanning string literal**错误。

字符串结尾忘记加引号了。仔细检查那行看看。

习题18 命名、变量、代码和函数

标题包含的内容够多的吧？接下来我要教你“函数”（function）了！说到函数，不同的人会有不一样的理解和使用方法，不过我只会教你现在能用到的最简单的使用方式。

函数可以做以下3件事情。

- 1.它们给代码段命名，就跟“变量”给字符串和数命名一样。
- 2.它们可以接收参数，就跟你的脚本接收argv一样。
- 3.使用#1和#2，它们可以让你创建“迷你脚本”或者“小命令”。

可以使用def新建函数。我将让你创建4个不同的函数，它们工作起来像你的脚本一样，然后我会演示各个函数之间的关系。

ex18.py

```
1  # this one is like your scripts with argv
2  def print_two(*args):
3      arg1, arg2 = args
4      print "arg1: %r, arg2: %r" % (arg1, arg2)
5
6  # ok, that *args is actually pointless, we can just do this
7  def print_two_again(arg1, arg2):
8      print "arg1: %r, arg2: %r" % (arg1, arg2)
9
10 # this just takes one argument
11 def print_one(arg1):
```

```
12     print "arg1: %r" % arg1
13
14 # this one takes no arguments
15 def print_none():
16     print "I got nothin'."
17
18
19 print_two("Zed","Shaw")
20 print_two_again("Zed","Shaw")
21 print_one("First!")
22 print_none()
```

让我们详解一下第一个函数 `print_two`，这个函数和你写脚本的方式差不多，因此看上去应该会觉着比较眼熟。

1.首先我们告诉Python使用`def`命令创建一个函数，也就是“定义”（`define`）的意思。

2.紧挨着`def`的是函数的名字。本例中它的名字是`print_two`，但名字可以随便取，叫`peanuts`也没关系，但最好函数名能够体现出函数的功能。

3.然后告诉函数，我们需要`*args` (`asterisk args`)，这和脚本的`argv`非常相似，参数必须放在圆括号（`()`）中才能正常工作。

4.接着用冒号（`:`）结束本行，然后开始下一行缩进。

5.冒号以下，使用4个空格缩进的行都是属于`print_two`这个函数的内容。其中第一行的作用是将参数解包，这和脚本参数解包的原理差不多。

6.为了演示它的工作原理，我们把解包后的每个参数都打印出来，这和我们在之前脚本练习中所做的类似。

函数`print_two`的问题是：它并不是创建函数最简单的方法。在

Python函数中，可以跳过整个参数解包的过程，直接使用()里边的名称作为变量名。这就是 `print_two_again` 实现的功能。

接下来的例子是`print_one`，它演示了函数如何接收单个参数。

最后一个例子是`print_none`，它演示了函数可以不接收任何参数。

警告 如果你不太能看懂上面的内容也别气馁，后面还有更多的习题展示如何创建和使用函数。现在你只要把函数理解成“迷你脚本”就可以了。

应该看到的结果

运行上面的脚本，会看到如下结果。

习题18 会话

```
$ python ex18.py
arg1: 'Zed', arg2: 'Shaw'
arg1: 'Zed', arg2: 'Shaw'
arg1: 'First!'
I got nothin'.
$
```

你应该已经了解函数是怎样工作的了。函数的用法和以前见过的 `exists`、`open` 及别的“命令”有点类似了吧？其实我只是为了让你容易理解才叫它们“命令”，它们其实本质上就是函数。也就是说，你也可以在自己的脚本中创建自己的“命令”。

附加练习

为自己写一个函数注意事项以供后续参考。你可以写在一个索引卡片上随时阅读，直到记住所有的要点为止。具体注意事项如下。

- 1.函数定义是以def开始的吗？
- 2.函数名是以字符和下划线_组成的吗？
- 3.函数名是不是紧跟着括号(？
- 4.括号里是否包含参数？多个参数是否以逗号隔开？
- 5.参数名称是否有重复？（不能使用重复的参数名。）
- 6.紧跟着参数的是不是括号和冒号（):）？
- 7.紧跟着函数定义的代码是否使用了4个空格的缩进？
- 8.函数结束的位置是否取消了缩进？

运行（使用、调用）一个函数时，记得检查下面的要点。

- 1.调用函数时是否使用了函数名？
- 2.函数名是否紧跟着(？
- 3.括号后有无参数？多个参数是否以逗号隔开？
- 4.函数是否以)结尾？

按照这两份检查表里的内容检查余下的习题，直到你不需要检查表为止。最后，将下面这句话读几遍：“运行（run）函数、调用（call）函数和使用（use）函数是同一个意思。”

常见问题回答

函数命名有什么规则？

和变量名一样，只要以字母、数字以及下划线组成，而且不是数字开始，就可以了。

***args**里的*是什么意思？

它的功能是告诉Python把函数的所有参数都接收进来，然后放到名叫args的列表中去。和一直在用的 `argv` 差不多，只不过前者是用在函数上。如果没什么特殊情况，我们一般不会经常用到这个东西。

这些任务好枯燥、好无聊啊。

你这么感觉就对了，说明你有了进步。你能明白代码的功用，而且写错代码的情况在你身上很少发生了。为了让任务不那么无聊，可以试着故意写错一些东西，看看会发生什么事情。

习题19 函数和变量

函数这个概念也许承载了太多的信息量，不过别担心。只要你坚持做这些习题，对照上一个习题中的检查表检查一遍这个习题，最终会明白这些内容的。

你可能没有注意到一个细节，我们现在强调一下：函数里的变量和脚本里的变量之间是没有联系的。下面的这个习题可以让你对这一点有更多的思考。

ex19.py

```
1 def cheese_and_crackers(cheese_count, boxes_of_crackers):
2     print "You have %d cheeses!" % cheese_count
3     print "You have %d boxes of crackers!" % boxes_of_crackers
4     print "Man that's enough for a party!"
5     print "Get a blanket.\n"
6
7
8 print "We can just give the function numbers directly:"
9 cheese_and_crackers(20, 30)
10
11
12 print "OR, we can use variables from our script:"
13 amount_of_cheese = 10
14 amount_of_crackers = 50
```

```
15
16 cheese_and_crackers(amount_of_cheese, amount_of_crackers)
17
18
19 print "We can even do math inside too:"
20 cheese_and_crackers(10 + 20, 5 + 6)
21
22
23 print "And we can combine the two, variables and math:"
24 cheese_and_crackers(amount_of_cheese      +      100,
amount_of_crackers + 1000)
```

以上代码展示了函数`cheese_and_crackers`的各种参数传递方式，函数会将传入的内容打印出来。我们可以直接给函数传递数字，也可以给它变量，还可以给它数学表达式，甚至可以把数学表达式和变量合起来用。

从一方面来说，函数的参数和生成变量时用的`=`赋值符类似。事实上，如果一个物件可以用`=`对其命名，通常也可以将其作为参数传递给一个函数。

应该看到的结果

你应该研究一下脚本的输出，和你想象的结果对比一下看有什么不同。

习题19 会话

```
$ python ex19.py
```

We can just give the function numbers directly:

You have 20 cheeses!

You have 30 boxes of crackers!

Man that's enough for a party!

Get a blanket.

OR, we can use variables from our script:

You have 10 cheeses!

You have 50 boxes of crackers!

Man that's enough for a party!

Get a blanket.

We can even do math inside too:

You have 30 cheeses!

You have 11 boxes of crackers!

Man that's enough for a party!

Get a blanket.

And we can combine the two, variables and math:

You have 110 cheeses!

You have 1050 boxes of crackers!

Man that's enough for a party!

Get a blanket.

附加练习

- 1.倒着将脚本读完，在每一行上面添加一行注释，说明这行的作用。
- 2.从最后一行开始，倒着阅读每一行，读出所有的重要字符来。
- 3.自己编写至少一个函数出来，然后用10种方式运行这个函数。

常见问题回答

怎么能有**10**种不同的方式运行一个函数呢？

信不信由你，理论上有无穷多种方式运行一个函数。在这里，试着按我在第8~12行给出的方式运行，当然你可以随意创新。

有没有办法可以分析这个函数的功能，以便我能理解？

有很多办法，最简单的一个办法是在每一行代码上面添加注释，另外一个办法是大声朗读代码，还有一个办法就是把代码打印出来，用笔画一些图示，并写一些注释说明。

怎样处理用户输入的数字，例如我想让用户输入**cracker**和**cheese**的数量？

记住，使用`int()`把`raw_input()`的值转换为整数。

第**13**行和第**14**行创建的变量会不会改变函数中的变量？

不会。这些变量是在函数之外的，当它们被传递到函数中以后，函数会为这些变量创建一些临时的版本，当函数运行结束后，这些临时变量就会被丢弃了，一切又回到了之前。继续阅读本书，后面你会更清楚这些概念。

把全局变量（如第**13**行和第**14**行）的名称和函数变量的名称取成一样的，这样做是不是不好？

是的，因为这样的话你就无法确定哪个是哪个了。有时候你可能会必须使用同一个变量名，有时候你会不小心使用了一样的变量名，不论如何，只要有可能，还是尽量避免变量的名称相同。

第**12**~**19**行是不是覆盖了函数**cheese_and_crackers**？

没有，完全没有。这只是函数调用而已。基本上就是这里会跳转到

函数的第一行，然后等函数运行完后再回到先前的位置，并没有用任何东西替换该函数。

函数的参数个数有限制吗？

取决于Python的版本和所用的操作系统，不过就算有限制，限值也是很大的。实际应用中，5个参数就不少了，再多就会让人头疼了。

可以在函数中调用函数吗？

可以。后面的习题中会用这一技巧写一个游戏。

习题20 函数和文件

回忆一下函数的要点，然后一边做这个习题，一边注意一下函数和文件是如何在一起协作发挥作用的。

ex20.py

```
1  from sys import argv
2
3  script, input_file = argv
4
5  def print_all(f):
6      print f.read()
7
8  def rewind(f):
9      f.seek(0)
10
11  def print_a_line(line_count, f):
12      print line_count, f.readline()
13
14  current_file = open(input_file)
15
16  print "First let's print the whole file:\n"
17
18  print_all(current_file)
```

```
19
20 print "Now let's rewind, kind of like a tape."
21
22 rewind(current_file)
23
24 print "Let's print three lines:"
25
26 current_line = 1
27 print_a_line(current_line, current_file)
28
29 current_line = current_line + 1
30 print_a_line(current_line, current_file)
31
32 current_line = current_line + 1
33 print_a_line(current_line, current_file)
```

特别注意一下，每次运行`print_a_line`时，我们是怎样传递当前的行号信息的。

应该看到的结果

习题20 会话

```
$ python ex20.py test.txt
```

```
First let's print the whole file:This is line 1
```

```
This is line 2
```

```
This is line 3
```

```
Now let's rewind, kind of like a tape.
```

```
Let's print three lines:
```

```
1 This is line 1
```

```
2 This is line 2
```

```
3 This is line 3
```

附加练习

- 1.通读脚本，在每一行之前加上注释，以理解脚本里发生的事情。
- 2.每次 `print_a_line` 运行时，你都传递了一个叫 `current_line` 的变量。每次调用函数时，打印出 `current_line` 的值，跟踪一下它在 `print_a_line` 中是怎样变成`line_count`的。
- 3.找出脚本中每一个用到函数的地方。检查`def`一行，确认参数没有用错。
- 4.上网研究一下`file`中的`seek`函数是做什么用的。试着运行`pydoc file`，看看能不能学到更多。
- 5.研究一下`+=`这个简写操作符的作用。写一个脚本，在里边用一下这个操作符。

常见问题回答

print_all和其他函数里的**f**是什么？

和习题18里的一样，**f**只是一个变量而已，不过在这里它指的是一个文件。Python里的文件就和老式磁带机或者DVD播放机差不多。它有一个用来读取数据的“磁头”，你可以通过这个“磁头”来操作文件。每次运行**f.seek(0)**就回到了文件的开始，而运行**f.readline()**则会读取文件的一行，然后将“磁头”移动到**\n**后面。后面会有更详细的解释。

为什么文件里会有间隔空行？

readline()函数返回的内容中包含文件本来就有的**\n**，而 **print** 在打印时又会添加一个**\n**，这样一来就会多出一个空行了。解决方法是在 **print** 语句结尾加一个逗号(,)，这样**print**就不会把它自己的**\n**打印出来了。

为什么**seek(0)**没有把**current_line**设为**0**？

首先**seek()**函数的处理对象是字节而非行，所以**seek(0)**只是转到文件的**0 byte**（也就是第一个字节）的位置。其次，**current_line**只是一个独立变量，和文件本身没有任何关系，我们只能手动为其增值。

+=是什么？

英语里边**it is**可以写成**it's**，**you are**可以写成**you're**，这叫做简写。而**+=**这个操作符是把**=**和**+**简写到一起了。**x += y**的意思和**x = x + y**是一样的。

readline()是怎么知道每一行在哪里的？

readline()里边的代码会扫描文件的每一个字节，直到找到一个**\n**为止，然后它停止读取文件，并且返回此前的文件内容。文件**f**会记录每次调用**readline()**后的读取位置，这样它就可以在下次被调用时读取接下

来的一行了。

习题21 函数可以返回某些东西

你已经学过使用=给变量命名，以及将变量定义为某个数字或者字符串。接下来我们将让你见证更多奇迹。我们要演示的是如何使用=以及一个新的Python词汇return来将变量设置为“一个函数的值”。有一点需要特别注意，不过我们暂且不讲，先写下面的脚本吧。

ex21.py

```
1  def add(a, b):
2      print "ADDING %d + %d" % (a, b)
3      return a + b
4
5  def subtract(a, b):
6      print "SUBTRACTING %d - %d" % (a, b)
7      return a - b
8
9  def multiply(a, b):
10     print "MULTIPLYING %d * %d" % (a, b)
11     return a * b
12
13  def divide(a, b):
14     print "DIVIDING %d / %d" % (a, b)
15     return a / b
16
```

```
17
18 print "Let's do some math with just functions!"
19
20 age = add(30, 5)
21 height = subtract(78, 4)
22 weight = multiply(90, 2)
23 iq = divide(100, 2)
24
25 print "Age: %d, Height: %d, Weight: %d, IQ: %d" % (age, height,
weight, iq
26
27
28 # A puzzle for the extra credit, type it in anyway.
29 print "Here is a puzzle."
30
31 what = add(age, subtract(height, multiply(weight, divide(iq, 2))))
32
33 print "That becomes: ", what, "Can you do it by hand?"
```

现在我们创建了自己的加、减、乘、除数学函数，即add、subtract、multiply和divide。重要的是函数的最后一行，如add的最后一行是return a + b，它实现以下几项功能。

- 1.我们调用函数时使用了两个参数，即a和b。
- 2.我们打印出这个函数的功能，这里就是计算加法。
- 3.接下来我们让Python做某个回传的动作：我们返回a + b的值。或者可以这么说：“我将a和b加起来，再把结果返回。”
- 4.Python将两个数字相加，然后当函数结束的时候，它就可以将a + b的结果赋予一个变量。

和本书里的很多其他东西一样，你要慢慢消化这些内容，一步一步执行下去，追踪一下究竟发生了什么。为了帮助你理解，本节的附加练习将让你解决一个谜题，并且让你学点儿比较酷的东西。

应该看到的结果

习题21 会话

```
$ python ex21.py
```

Let's do some math with just functions!

ADDING 30 + 5

SUBTRACTING 78 - 4

MULTIPLYING 90 * 2

DIVIDING 100 / 2

Age: 35, Height: 74, Weight: 180, IQ: 50

Here is a puzzle.

DIVIDING 50 / 2

MULTIPLYING 180 * 25

SUBTRACTING 74 - 4500

ADDING 35 + -4426

That becomes: -4391 Can you do it by hand?

附加练习

1.如果你不是很确定`return`的功能，试着自己写几个函数，让它们返回一些值。你可以将任何可以放在`=`右边的东西作为一个函数的返回值。

2.这个脚本的结尾是一个谜题。我将一个函数的返回值用作了另外一个函数的参数。我将它们链接到了一起，就跟写数学等式一样。这样可能有些难读，不过运行一下你就知道结果了。接下来，你需要试试看能不能用正常的方法实现和这个表达式一样的功能。

3.一旦你解决了这个谜题，试着修改一下函数里的某些部分，然后看会有什么样的结果。你可以有目的地修改它，让它输出另外一个值。

4.最后，颠倒过来做一次。写一个简单的等式，使用一样的函数来计算它。

这个习题可能会让你有些头大，不过还是慢慢来，把它当做一个游戏，解决这样的谜题正是编程的乐趣之一。后面还会有类似的谜题。

常见问题回答

为什么**Python**会把函数或公式倒着打印出来？

其实不是倒着打印，而是自内而外打印。如果你把函数内容逐句看下去，你会发现其中的规律。试着搞清楚为什么说它是“自内而外”而不是“倒着”。

怎样使用**raw_input()**输入自定义值？

记得 `int(raw_input())`吧？不过这样也有一个问题，那就是无法输入浮点数，所以可以试着使用**float(raw_input())**。

你说的“写一个公式”是什么意思？

来个简单的例子： $24 + 34 / 100 - 1023$ ——把它转换成函数的形式。然后自己想一些数学式子，像公式一样用变量写出来。

习题22 到现在你学到了哪些东西

这个习题以及下一个习题中不会有任何代码，所以也不会有“应该看到的结果”或者“附加练习”。其实这个习题可以说是一个大的附加练习。我将让你完成一个表格，回顾一下到现在为止已经学到的所有知识。

首先，回到每一个习题的脚本里，把你遇到的每一个词和每一个符号（字符的别名）写下来。确保你的符号列表是完整的。

下一步，在每一个关键字和符号后面写出它的名字，并说明它的作用。如果你在书里找不到符号的名字，就上网找一下。如果你不知道某个关键字或者符号的作用，就回到用到该关键字或者符号的习题通读一下，并且在脚本中测试一下它们的用处。

你也许会遇到一些如论如何找不到答案的东西，把这些记在列表里，它可以提示你还有哪些东西自己不懂，等下次遇到的时候，你就不会轻易跳过了。

你的列表做好以后，再花几天时间重写一遍这份列表，确认里边的东西都是正确的。你可能觉得这很无聊，不过你还是需要坚持完成任务。

等你记住了这份列表中的所有内容，就试着把这份列表默写一遍。如果发现自己漏掉或者忘记了某些内容，就回去再记一遍。

警告 做这个习题最重要的一点是：“没有失败，只有尝试。”

学到的东西

这种记忆练习是枯燥无味的，所以知道它的意义很重要。它会让你明确目标，让你知道自己所有努力的目的。

在这个习题中你学会的是各种符号的名称，这样读代码对你来说会更加容易。这和学英语时记字母表和基本单词的意思是一样的，不同的是Python中会有一些你不熟悉的字符。

慢慢做，别让它成为负担。这些符号对你来说应该比较熟悉，所以记住它们应该不是很费力。你可以一次花个15分钟，然后休息一下。劳逸结合可以学得更快，而且可以保持士气。

习题23 阅读一些代码

上一周你应该已经牢记了你的符号列表。现在你需要将这些运用起来，再花一周的时间，在网上阅读代码。这个任务初看会觉得很艰巨。我将直接把你丢到深水区呆几天，让你竭尽全力去读懂实实在在的项目里的代码。这个习题的目的不是让你读懂，而是让你学会下面的技能。

- 1.找到你需要的Python代码。
- 2.通读代码，找到文件。
- 3.尝试理解你找到的代码。

以现在的水平，你还不具备完全理解你找到的代码的能力，不过通过接触这些代码，你可以熟悉真正的编程项目是什么样子。

做这个习题时，你可以把自己当成是一个人类学家来到了一片陌生的大陆，你只懂一丁点本地语言，但你需要接触当地人并且生存下去。当然做习题不会遇到生存问题，毕竟这不是在荒野或者丛林。

你要做的事情具体如下。

- 1.使用浏览器登录bitbucket.org，搜索“python”。
- 2.忽略那些提到“Python 3”的项目，它们只会让你变迷糊。
- 3.随便找一个项目，然后点进去。
- 4.点击 **Source** 标签，浏览目录和文件列表，直到看到以.py 结尾的文件（`setup.py`就别看了，这样的文件看了也没用）。
- 5.从头开始阅读你找到的代码，把它的功能用笔记记下来。
- 6.如果看到一些有趣的符号或者奇怪的单词，你可以把它们记下来，日后再进行研究。

就是这样，你的任务是使用目前学到的知识，看自己能不能读懂一些代码，看出它们的功能来。你可以先粗略地阅读，然后再细读。也许你还可以试试将难度比较大的部分一字不漏地朗读出来。

现在再试试下面这几个站点。

launchpad.net

sourceforge.net

freecode.com

习题24 更多练习

现在离本书第一部分的结尾已经不远了，你应该已经具备了足够的Python基础知识，可以继续学习一些编程的原理了，但你应该做更多的练习。这个习题的内容比较长，它的目的是锻炼你的毅力，下一个习题也差不多是这样的，好好完成它们，做到完全正确，记得仔细检查。

ex24.py

```
1  print "Let's practice everything."
2  print 'You\'d need to know \'bout escapes with \\ that do \n newlines
and \t tabs.'
3
4  poem = """
5  \tThe lovely world
6  with logic so firmly planted
7  cannot discern \n the needs of love
8  nor comprehend passion from intuition
9  and requires an explanation
10 \n\t\twhere there is none.
11  """
12
13 print "-----"
14 print poem
15 print "-----"
```

```
16
17
18 five = 10 - 2 + 3 - 6
19 print "This should be five: %s" % five
20
21 def secret_formula(started):
22     jelly_beans = started * 500
23     jars = jelly_beans / 1000
24     crates = jars / 100
25     return jelly_beans, jars, crates
26
27
28 start_point = 10000
29 beans, jars, crates = secret_formula(start_point)
30
31 print "With a starting point of: %d" % start_point
32 print "We'd have %d beans, %d jars, and %d crates." % (beans,
jars, crates)
33
34 start_point = start_point / 10
35
36 print "We can also do that this way:"
37 print "We'd have %d beans, %d jars, and %d crates." %
secret_formula(start_point)
```

应该看到的结果

习题24 会话

```
$ python ex24.py
```

```
Let's practice everything.
```

```
You'd need to know 'bout escapes with \ that do newlines and tabs.
```

```
-----
```

```
        The lovely world
with logic so firmly planted
cannot discern
    the needs of love
nor comprehend passion from intuition
and requires an explanation
        where there is none.
```

```
-----
```

```
This should be five: 5
```

```
With a starting point of: 10000
```

```
We'd have 5000000 beans, 5000 jars, and 50 crates.
```

```
We can also do that this way:
```

```
We'd have 500000 beans, 500 jars, and 5 crates.
```

附加练习

- 1.记得仔细检查结果，从后往前倒着检查，把代码朗读出来，在不清楚的位置加上注释。
- 2.故意把代码改错，运行并检查会发生什么样的错误，并且确认你有能力改正这些错误。

常见问题回答

为什么你在后面把**jelly_beans**这个变量名又叫成了**beans**?

这是函数的工作原理。记住函数内部的变量都是临时的，当你的函数返回以后，返回值可以被赋予一个变量。我这里是创建了一个新变量，用来存放函数的返回值。

倒着读代码是什么意思?

从最后一行开始，把你写的代码和我写的代码进行比较。如果这一行完全一样，就接着比较上一行，直到全部比较完为止。

这首诗是谁写的?

我写的。我的诗也还可以吧。

习题25 更多更多的实践

我们将做一些关于函数和变量的练习，以确认你真正掌握了这些知识。这个习题对你来说应该很简单：写程序，逐行研究，弄懂它。

不过这个习题还是有些不同，你不需要运行它，取而代之，你将它导入到Python里，并自己运行这些函数。

ex25.py

```
1  def break_words(stuff):
2      """This function will break up words for us."""
3      words = stuff.split(' ')
4      return words
5
6  def sort_words(words):
7      """Sorts the words."""
8      return sorted(words)
9
10 def print_first_word(words):
11     """Prints the first word after popping it off."""
12     word = words.pop(0)
13     print word
14
15 def print_last_word(words):
16     """Prints the last word after popping it off."""
```



```
17     word = words.pop(-1)
18     print word
19
20 def sort_sentence(sentence):
21     """Takes in a full sentence and returns the sorted words."""
22     words = break_words(sentence)
23     return sort_words(words)
24
25 def print_first_and_last(sentence):
26     """Prints the first and last words of the sentence."""
27     words = break_words(sentence)
28     print_first_word(words)
29     print_last_word(words)
30
31 def print_first_and_last_sorted(sentence):
32     """Sorts the words then prints the first and last one."""
33     words = sort_sentence(sentence)
34     print_first_word(words)
35     print_last_word(words)
```

首先以正常的方式python ex25.py运行，找出你犯的错误，并把它们都改正过来，然后你需要跟着下面的“应该看到的结果”完成这个习题。

应该看到的结果

这个习题将在你之前用来做算术的 Python 解释器里，用交互的方式和你的.py 交流。在shell里像下面这样运行它：

```
$ python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc.build 5658)] (LLVM build
2335.15.00) on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

你看到的可能和我的有一点儿不同，但是一旦你看到>>>提示符，你就可以录入代码并立即运行它了。

下面是我做的时候看到的样子。

习题25 会话

```
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc.build 5658)] (LLVM build
2335.15.00) on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import ex25
>>> sentence = "All good things come to those who wait."
>>> words = ex25.break_words(sentence)
>>> words
['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
>>> sorted_words = ex25.sort_words(words)
```

```
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_word(words)
All
>>> ex25.print_last_word(words)
wait.
>>> wrods
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'wrods' is not defined
>>> words
['good', 'things', 'come', 'to', 'those', 'who']
>>> ex25.print_first_word(sorted_words)
All
>>> ex25.print_last_word(sorted_words)
who
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait.']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
>>> ex25.print_first_and_last(sentence)
All
wait.
>>> ex25.print_first_and_last_sorted(sentence)
All
who
```

我们来逐行分析一下每一步实现的是什麼。

在第5行将ex25.py执行了import，和已经做过的其他import一样。在import的时候不需要加.py后缀。这个过程里，把ex25.py当做了个“模块”来使用，在这个模块里定义的函数也可以直接调用出来。

第6行创建了一个用来处理的“语句”。

第7行使用ex25调用你的第一个函数ex25.break_words。其中的.符号可以告诉Python：“嗨，我要运行ex25里那个叫break_words的函数！”

第8行只是输入words，而Python将在第9行打印出这个变量里有什么。看上去可能会觉得奇怪，不过这其实是一个“列表”，会在后面的章节中讲到。

第10~11行使用ex25.sort_words来得到一个排序过的句子。

第13~16行使用ex25.print_first_word和ex25.print_last_word将第一个和最后一个词打印出来。

第17行比较有趣。我把words变量写错成了wrods，所以Python给出一条关于第18~20行的错误信息。

第21~22行打印出修改过的词汇列表。第一个和最后一个单词已经打印过了，所以在这里没有再次打印出来。

剩下的行需要你自已分析，就留作附加练习了。

附加练习

1.研究答案中没有分析过的行，找出它们的来龙去脉。确认自己明白了使用的是模块ex25中定义的函数。

2.试着执行`help(ex25)`和`help(ex25.break_words)`。这是得到模块帮助文档的方式。所谓帮助文档就是定义函数时放在`"""`之间的东西，它们也被称作文档注释（documentation comment），后面还会出现更多类似的东西。

3.重复键入ex25.是很烦的一件事情。有一个捷径就是用`from ex25 import *`的方式导入模块。这相当于说：“我要把ex25中所有的东西导入进来。”程序员喜欢说这样的倒装句，开一个新的会话，看看所有的函数是不是已经在那里了。

4.试着将代码文件分解，看看Python使用你的代码文件时是怎样的状况。如果要重新加载代码文件，你需要先用`Ctrl+D`（Windows下用`Ctrl+Z`）来退出Python。

常见问题回答

有的函数打印出来的结果是**None**。

也许你的函数漏写了最后的**return**语句。回到代码中检查一下是不是每一行都写对了。

输入**import ex15**时显示**-bash: import: command not found**。

注意看“应该看到的结果”部分。我是在Python中写的这句，不是在终端直接写的。你要先运行python再输入代码。

输入**import ex25.py**时显示**ImportError: No module named ex25.py**。

.py是不需要的。Python知道文件是.py结尾，所以只要输入**import ex25**即可。

运行时提示**SyntaxError: invalid syntax**。

这说明你在提示的那行有一个语法错误，可能是漏了半个括号或者引号，也可能是别的。一旦看到这种错误，应该去对应的行检查代码，如果那一行没问题，就倒着继续往上检查每一行，直到发现问题为止。

函数里的代码不是只在函数里有效吗？为什么**words.pop(0)**这个函数会改变**words**的内容？

这个问题有点儿复杂，不过在这里 **words** 是一个列表，可以对它进行操作，操作结果也可以被保存下来。这和操作文件**f.readline()**时的工作原理差不多。

函数里什么时候该用**print**，什么时候该用**return**？

print只是屏幕输出而已，你可以让一个函数既**print**又**return**值。明白这一点后，你就知道这个问题其实没什么意义。如果你想要打印到屏

幕，那就使用`print`；如果是想返回值，那就是用`return`。

习题26 恭喜你，现在可以考试了！

现在已经差不多完成这本书的前半部分了，不过后半部分才是更有趣的。你将学到逻辑，并通过条件判断实现有用的功能。

在继续学习之前，有一道试题要你做。这道试题很难，因为它需要你修正别人写的代码。你做程序员以后，需要经常面对别的程序员的代码，也许还要面对他们的傲慢态度，他们会经常说自己的代码是完美的。

这样的程序员是自以为是、不在乎别人的。真正优秀的科学家会对他们自己的工作持怀疑态度，同样，真正优秀的程序员也会认为自己的代码总有出错的可能，他们会先假设是自己的代码有问题，然后用排除法清查所有可能是自己有问题的地方，最后才会得出“这是别人的错误”这样的结论。

在这个习题中，你将面对一个水平很糟糕的程序员，并改好他的代码。我将习题 24 和习题25胡乱复制到了一个文件中，随机地删掉了一些字符，然后添加了一些错误进去。大部分的错误是Python在执行时会告诉你的，还有一些算术错误是你要自己找出来的，再剩下下来的就是格式和拼写错误了。

所有这些错误都是程序员很容易犯的，就算有经验的程序员也不例外。

你的任务是将此文件修改正确，用你所有的技能改进这个脚本。你可以先分析这个文件，或者你还可以把它像学期论文一样打印出来，修正里边的每一个缺陷，重复修正和运行的动作，直到这个脚本可以完美

地运行起来。在整个过程中不要寻求帮助，如果卡在某个地方无法进行下去，那就休息一会晚点儿再做。

就算你需要几天才能完成，也不要放弃，直到完全改对为止。

最后要说的是，这个习题的目的不是写程序，而是修正现有的程序，你需要访问网站<http://learnpythonthehardway.org/exercise26.txt>，从那里把代码复制粘贴过来，命名为 `ex26.py`，这也是本书唯一一处允许你复制粘贴的地方。

常见问题回答

一定要**import ex25.py**吗？移除对它的引用也可以吧？

怎样都可以。不过这个文件里会用到ex25中的函数，你可以试着移除引用看看会怎样。

我可以边修正代码边运行吗？

当然可以。这样的事情就是要计算机帮忙，多多益善。

习题27 记住逻辑关系

到此为止，你已经学会了读写终端文件和很多Python数学运算功能。现在，你要开始学习逻辑了。你要学习的不是研究院里的高深逻辑理论，只是程序员每天都用到的让程序跑起来的基础逻辑知识。

学习逻辑之前你需要先记住一些东西。这个习题你要在一个星期内完成，不要擅自修改日程，就算你烦得不得了，也要坚持下去。这个习题会让你背下来一系列的逻辑表格，这样完成后面的习题更容易。

需要事先警告你的是：这件事情一开始一点乐趣都没有，你一开始就会觉得它很无聊乏味，但它的目的是教你程序员必需的一个重要技能——一些重要的概念是必须记住的，一旦你明白了这些概念，就会获得相当的成就感，但是一开始你会觉得它们很难掌握，但等到某一天，你会刷地一下豁然开朗。你会从这些基础的学习中得到丰厚的回报。

这里告诉你一个记住某样东西，而不让自己抓狂的方法：在一整天里，每次记忆一小部分，把你最需要加强的部分标记起来。不要想着在两小时内连续不停地背，这不会有什么好的结果。不管你花多长时间，你的大脑也只会留住你在前15分钟或者前30分钟内看过的东西。

取而代之，你要做的是创建一些索引卡片，卡片有两列内容，正面写下逻辑关系，反面写下答案。你需要达到的效果是：拿出一张卡片来，看到正面的表达式，如“True or False”，可以立即说出背面的结果是“True”。坚持练习，直到能做到这一点为止。

一旦能做到这一点了，接下来你需要每天晚上自己在笔记本上写一份真值表出来。不要只是抄写这张表，试着默写这张表，如果发现哪里

没记住，就飞快地撇一眼答案。这样会训练你的大脑，让它记住整个真值表。

不要在这上面花超过一周的时间，因为你在后面的应用过程中还会继续学习它们。

逻辑术语

在Python中会使用下面的术语（字符或者词汇）来定义事物的真（True）或者假（False）。计算机的逻辑就是在程序的某个位置检查这些字符或者变量组合在一起表达的结果是真是假。

and	与
or	或
not	非
!=	不等于
==	等于
>=	大于等于
<=	小于等于
True	真
False	假

这些字符其实你已经见过了，但这些单词你可能还没见过。这些单词（and、or和not）和你期望的效果其实是一样的，跟英语里的意思一模一样。

真值表

我们将使用这些字符来创建你需要记住的真值表。

NOT	True?
not False	True
not True	False

OR	True?
True or False	True
True or True	True
False or True	True
False or False	False

AND	True?
True and False	False
True and True	True
False and True	False
False and False	False

NOT OR	True?
not (True or False)	False
not (True or True)	False
not (False or True)	False
not (False or False)	True

NOT AND	True?
not (True and False)	True
not (True and True)	False
not (False and True)	True
not (False and False)	True

!=	True?
1 != 0	True
1 != 1	False
0 != 1	True
0 != 0	False

==	True?
1 == 0	False
1 == 1	True
0 == 1	False
0 == 0	True

现在使用这些表格创建你自己的卡片，再花一个星期慢慢记住它们。记住一点，每天尽力去学，在尽力的基础上多花一点功夫。

常见问题回答

直接学习布尔算法，不用背这些东西，可不可以？

当然可以，不过这么一来，当你写代码的时候，你就需要回头想布尔函数的原理，这样写代码的速度就慢了。如果你记下来了，不但锻炼了自己的记忆力，而且让这些应用变成了条件反射，理解起来就更容易了。当然，你觉得哪种方法好，就用哪种方法好了。

习题28 布尔表达式练习

上一节的逻辑组合的正式名称是“布尔逻辑表达式”（boolean logic expression）。在编程中，布尔逻辑可以说是无处不在。它们是计算机运算的基础和重要组成部分，掌握它们就跟学音乐掌握音阶一样重要。

在这个习题中，将在Python里使用前一个习题中学到的逻辑表达式。先为下面的每一个逻辑问题写出你的答案，每一题的答案要么为True要么为False。写完以后，将Python运行起来，输入这些逻辑语句，确认你写的答案是否正确。

1. True and True
2. False and True
3. 1==1 and 2 == 1
4. "test" == "test"
5. 1== 1 or 2 != 1
6. True and 1 == 1
7. False and 0 != 0
8. True or 1 == 1
9. "test" == "testing"
10. 1 != 0 and 2 == 1
11. "test" != "testing"
12. "test" == 1
13. not (True and False)
14. not (1 == 1 and 0 != 1)

15. `not (10 == 1 or 1000 == 1000)`
16. `not (1 != 10 or 3 == 4)`
17. `not ("testing" == "testing" and "Zed" == "Cool Guy")`
18. `1 == 1 and not ("testing" == 1 or 1 == 0)`
19. `"chunky" == "bacon" and not (3 == 4 or 3 == 3)`
20. `3 == 3 and not ("testing" == "testing" or "Python" == "Fun")`

在本节结尾的地方我会给你一种理清复杂逻辑的技巧。

所有的布尔逻辑表达式都可以用下面的简单流程得到结果。

1.找到相等判断的部分（`==` or `!=`），将其改写为其最终值（`True`或`False`）。

2.找到括号里的`and/or`，先算出它们的值。

3.找到每一个`not`，算出它们取反的值。

4.找到剩下的`and/or`，解出它们的值。

5.都做完后，剩下的结果应该就是`True`或者`False`了。

下面以第20个逻辑表达式演示一下：

`3 != 4 and not ("testing" != "test" or "Python" == "Python")`

接下来你将看到这个复杂的表达式是如何被逐级解为一个结果的。

1.解出每一个等值判断。

a.`3 != 4`为`True`: `True and not ("testing" != "test" or "Python"=="Python")`

b.`"testing" != "test"` 为`True`: `True and not (True or "Python"=="Python")`

c.`"Python" == "Python"`: `True and not (True or True)`

2.找到括号中的每一个`and/or`。

`(True or True)`为`True`: `True and not (True)`

3.找到每一个`not`并将其取反。

`not (True)`为`False`: `True and False`

4.找到剩下的and/or，解出它们的值。

True and False为False

这样我们就解出了它最终的值为False。

警告 复杂的逻辑表达式一开始看上去可能会让你觉得很难。你也许已经碰壁过了，不过别灰心，这些“逻辑体操”式的训练只是让你逐渐习惯起来，以便后面你可以轻易应对编程里边更酷的一些东西。只要坚持下去，不放过自己做错的地方就行了。如果你暂时不太能理解也没关系，最终总是会弄懂的。

应该看到的结果

下面是通过与Python对话得到的结果。

```
$ python
```

```
Python 2.5.1 (r251:54863, Feb 6 2009, 19:02:12)
```

```
[GCC 4.0.1 (Apple Inc.build 5465)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> True and True
```

```
True
```

```
>>> 1 == 1 and 2 == 2
```

```
True
```

附加练习

1. Python里还有很多和!=、==类似的操作符。试着尽可能多地列出Python中的等价运算符，如<或者<=。
2. 写出每一个等价运算符的名称，如!=叫“不等于”。
3. 在Python中测试新的布尔操作。在按回车键前你需要说出它的结果。不要思考，凭自己的第一感觉就可以了。把表达式和结果用笔写下来再按回车键，最后看自己做对多少，做错多少。
4. 把习题3那张纸丢掉，以后你再也不需要查它了。

常见问题回答

为什么 **"test" and "test"** 返回 **"test"**，**1 and 1** 返回 **1**，而不是返回 **True** 呢？

Python和很多语言一样，都是返回两个被操作对象中的一个，而非它们的布尔表达式True或 False。这意味着，如果你写了 **False and 1**，得到的是第一个操作数（False），而非第二个操作数（1）。多做几个实验。

!=和**<>**有何不同？

Python中**!=**是主流用法，**<>**将被逐渐废弃，除此以外没什么不同。

有没有短路逻辑？

有的。任何以False开头的and语句都会直接处理成False，不会继续检查后面语句。任何包含True的or语句，只要处理到True，就不会继续向下推算，而是直接返回True了。不过，还是要确保整个语句都能正常处理，以方便日后理解和使用代码。

习题29 if语句

下面是要完成的作业，介绍了if语句。输入这段代码，让它能正确运行，然后看看你是否有所收获。

ex29.py

```
1  people = 20
2  cats = 30
3  dogs = 15
4
5
6  if people < cats:
7      print "Too many cats! The world is doomed!"
8
9  if people > cats:
10     print "Not many cats! The world is saved!"
11
12 if people < dogs:
13     print "The world is drooled on!"
14
15 if people > dogs:
16     print "The world is dry!"
17
18
```



```
19  dogs += 5
20
21  if people >= dogs:
22      print "People are greater than or equal to dogs."
23
24  if people <= dogs:
25      print "People are less than or equal to dogs."
26
27
28  if people == dogs:
29      print "People are dogs."
```

应该看到的结果

习题29 会话

```
$ python ex29.py
```

```
Too many cats! The world is doomed!
```

```
The world is dry!
```

```
People are greater than or equal to dogs.
```

```
People are less than or equal to dogs.
```

```
People are dogs.
```

附加练习

猜猜 if 语句是什么，它有什么用处。在做下一道习题前，试着用自己的话回答一下下面的问题。

- 1.你认为if对它的下一行代码做了什么？
- 2.为什么if语句的下一行需要4个空格的缩进？
- 3.如果不缩进会发生什么事情？
- 4.把习题27中的其他布尔表达式放到if语句中会不会也可以运行呢？试一下。
- 5.如果把变量people、cats和dogs的初始值改掉会发生什么事情？

常见问题回答

`+=`是什么意思？

`x += 1`和`x = x + 1`一样，只不过可以少打几个字母。你可以把它叫做“递增”运算符。你后面还会学到`-=`以及很多别的表达式。

习题30 else和if

前一个习题中写了一些if语句，并且试图猜出它们是什么，以及实现的是什么功能。在继续学习之前，我解释一下上一个习题中附加练习的答案。上一个习题的附加练习你做过了吧？

1.你认为if对它的下一行代码做了什么？if语句为代码创建了一个所谓的“分支”，就跟RPG游戏中的情节分支一样。if语句告诉你的脚本：如果这个布尔表达式为真，就运行接下来的代码，否则就跳过这一段。

2.为什么if语句的下一行需要4个空格的缩进？行尾的冒号的作用是告诉Python接下来你要创建一个新的代码块。这跟你创建函数时的冒号是一个道理。

3.如果不缩进会发生什么事情？如果没有缩进，你应该会看到Python 报错。Python 的规则里，只要一行以冒号（:）结尾，它接下来的内容就应该有缩进。

4.把习题27中的其他布尔表达式放到if语句中会不会也可以运行呢？试一下。可以，而且不管多复杂都可以，虽然写复杂的东西通常是一种不好的编程风格。

5.如果把变量 people、cats 和 dogs 的初始值改掉会发生什么事情？因为你比较的对象是数字，所以，如果把这些数字改掉的话，某些位置的if语句会被演绎为True，而它下面的代码块将被运行。你可以试着修改这些数字，然后在头脑里假想一下哪一段代码会被运行。

把我的答案和你的答案比较一下，确认自己真正弄懂了“代码块”的含义。因为下一个习题将会写很多的if语句，所以这一点对于做下一个

习题很重要。

把下面这段写下来，并让它运行起来。

ex30.py

```
1  people = 30
2  cars = 40
3  buses = 15
4
5
6  if cars > people:
7      print "We should take the cars."
8  elif cars < people:
9      print "We should not take the cars."
10 else:
11     print "We can't decide."
12
13 if buses > cars:
14     print "That's too many buses."
15 elif buses < cars:
16     print "Maybe we could take the buses."
17 else:
18     print "We still can't decide."
19
20 if people > buses:
21     print "Alright, let's just take the buses."
22 else:
23     print "Fine, let's stay home then."
```

应该看到的结果

习题30 会话

```
$ python ex30.py
```

```
We should take the cars.
```

```
Maybe we could take the buses.
```

```
Alright, let's just take the buses.
```

附加练习

- 1.猜想一下elif和else的功能。
- 2.将cars、people和buses的数改掉，然后追溯每一个if语句。看看最后会打印出什么。
- 3.试着写一些复杂的布尔表达式，如cars > people and buses < cars。
- 4.在每一行的上面加上注释，说明这一行的作用。

常见问题回答

如果多个**elif**块都是**True**，**Python**会如何处理？

Python只会运行它遇到的是**True**的第一个块，所以只有第一个为**True**的块会运行。

习题31 作出决定

在这本书的上半部分，你打印了一些东西，而且调用了函数，不过一切都是线性进行的。你的脚本从最上面一行开始，一路运行到结束，但其中并没有决定程序流向的分支点。现在已经学了if、else和elif，可以开始创建包含条件判断的脚本了。

上一个脚本中你写了一系列的简单提问测试。这个习题的脚本中，你需要向用户提问，依据用户的答案来做出决定。把脚本写下来，多鼓捣一阵子，看看它的工作原理是什么。

ex31.py

```
1  print "You enter a dark room with two doors.  Do you go through
door #1 or door #2?"
2
3  door = raw_input("> ")
4
5  if door == "1":
6      print "There's a giant bear here eating a cheese cake.  What do
you do?"
7      print "1.Take the cake."
8      print "2.Scream at the bear."
9
10     bear = raw_input("> ")
11
```

```

12         if bear == "1":
13             print "The bear eats your face off.  Good job!"
14         elif bear == "2":
15             print "The bear eats your legs off.  Good job!"
16         else:
17             print "Well, doing %s is probably better.  Bear runs
away." % bear
18
19     elif door == "2":
20         print "You stare into the endless abyss at Cthulhu's retina."
21         print "1.Blueberries."
22         print "2.Yellow jacket clothespins."
23         print "3.Understanding revolvers yelling melodies."
24
25         insanity = raw_input("> ")
26
27         if insanity == "1" or insanity == "2":
28             print "Your body survives powered by a mind of jello.
Good job!"
29         else:
30             print "The insanity rots your eyes into a pool of muck.
Good job!"
31
32     else:
33         print "You stumble around and fall on a knife and die.  Good
job!"

```

这里的重点是你可以在if语句内部再放一个if语句。这是一个很强

大的功能，可以用来创建嵌套的决定，其中的一个分支将引向另一个分支的子分支。

你需要理解if语句包含if语句的概念。做一下附加练习，确信自己真正理解了它们。

应该看到的结果

我在玩一个小冒险游戏，我玩的水平不怎么好。

习题31 会话

```
$ python ex31.py
```

```
You enter a dark room with two doors.Do you go through door #1 or  
door #2?
```

```
> 1
```

```
There's a giant bear here eating a cheese cake.What do you do?
```

```
1.Take the cake.
```

```
2.Scream at the bear.
```

```
> 2
```

```
The bear eats your legs off.Good job!
```

附加练习

为游戏添加新的部分，改变玩家做决定的位置。尽自己的能力扩展这个游戏，不过别把游戏弄得太怪异了。

常见问题回答

可以用多个**if/else**来取代**elif**吗？

有时候可以，不过这也取决于if/else是怎样写的，而且这样一来Python就需要去检查每一处if/else，而不是像if/elif/else一样，只要检查到第一个True就可以停下来了。试着写些代码看两者有何不同。

怎样判断一个数处于某个值域中？

两种办法：经典语法是使用 $1 < x < 10$ ，用`x in range(1, 10)`也可以。

怎样用**if/elif/else**块实现4个以上的条件判断？

简单，多写几个elif块就可以了。

习题32 循环和列表

现在你应该有能力写更有趣的程序出来了。如果你能一直跟得上，你应该已经看出将if语句和“布尔表达式”结合起来可以让程序作出一些智能化的事情。

然而，我们的程序还需要能很快地完成重复的事情。这个习题中我们将使用for循环来创建和打印出各种各样的列表。在做的过程中，你会逐渐明白它们是怎么回事。现在我不会告诉你，你需要自己找到答案。

在你开始使用for循环之前，你需要在某个位置存放循环的结果。最好的方法是使用列表(list)，顾名思义，它就是一个按顺序存放东西的容器。列表并不复杂，你只是要学习一点新的语法。首先我们看看如何创建列表：

```
hairs = ['brown', 'blond', 'red']
eyes = ['brown', 'blue', 'green']
weights = [1, 2, 3, 4]
```

你要做的是以左方括号（[）开头“打开”列表，然后写下你要放入列表的东西，用逗号隔开，就跟函数的参数一样，最后你需要用右方括号（]）结束右方括号的定义。然后 Python接收这个列表以及里边所有的内容，将其赋给一个变量。

警告 对于不会编程的人来说这是一个难点。习惯性思维告诉你的大脑大地是平的。记得上一个练习中的 if 语句嵌套吧，你可能觉得要理解它有些难度，因为生活中一般人不会去像这样的问题，但这样的问题在编程中几乎到处都是。你会看到一个函数调用另外一个包含 if 语句的

函数，其中又有嵌套列表的列表。如果你看到这样的东西一时无法弄懂，就用纸笔记下来，手动分割下去，直到弄懂为止。

现在我们将使用循环创建一些列表，然后将它们打印出来。

ex32.py

```
1  the_count = [1, 2, 3, 4, 5]
2  fruits = ['apples', 'oranges', 'pears', 'apricots']
3  change = [1, 'pennies', 2, 'dimes', 3, 'quarters']
4
5  # this first kind of for-loop goes through a list
6  for number in the_count:
7      print "This is count %d" % number
8
9  # same as above
10 for fruit in fruits:
11     print "A fruit of type: %s" % fruit
12
13 # also we can go through mixed lists too
14 # notice we have to use %r since we don't know what's in it
15 for i in change:
16     print "I got %r" % i
17
18 # we can also build lists, first start with an empty one
19 elements = []
20
21 # then use the range function to do 0 to 5 counts
22 for i in range(0, 6):
23     print "Adding %d to the list." % i
```

```
24     # append is a function that lists understand
25     elements.append(i)
26
27 # now we can print them out too
28 for i in elements:
29     print "Element was: %d" % i
```

应该看到的结果

习题32 会话

```
$ python ex32.py
This is count 1
This is count 2
This is count 3
This is count 4
This is count 5
A fruit of type: apples
A fruit of type: oranges
A fruit of type: pears
A fruit of type: apricots
I got 1
I got 'pennies'
I got 2
I got 'dimes'
I got 3
I got 'quarters'
Adding 0 to the list.
Adding 1 to the list.
Adding 2 to the list.
Adding 3 to the list.
Adding 4 to the list.
```

Adding 5 to the list.

Element was: 0

Element was: 1

Element was: 2

Element was: 3

Element was: 4

Element was: 5

附加练习

- 1.注意一下range的用法。查一下range函数并理解它。
- 2.在第22行，可以直接将elements赋值为range(0,6)，而无需使用for循环吗？
- 3.在 Python 文档中找到关于列表的内容，仔细阅读一下，除了append 以外列表还支持哪些操作？

常见问题回答

如何创建二维列表？

就是在列表中包含列表，如[[1,2,3],[4,5,6]]。

列表和数组不是一样的吗？

取决于语言和实现方式。从经典意义上理解的话，列表和数组是很不同的，因为它们的实现方式不同。在Ruby语言中列表和数组都被叫做数组，而在Python中又都叫做列表。现在我们就把它叫列表吧，因为Python里就是这么叫的。

为什么**for**循环可以使用未定义的变量？

循环开始时这个变量就被定义了，当然每次循环它都会被重新定义一次。

为什么**for i in range(1, 3):**只循环2次而非3次？

`range()`函数会从第一个数到最后一个数，但不包含最后一个数。所以，它到2的时候就停止了，而不会到3。这种含首不含尾的方式是循环中极其常见的一种用法。

elements.append()是什么功能？

它的功能是在列表的尾部追加元素。打开Python命令行，创建几个列表试验一下。以后每次遇到自己不明白的东西，你都可以在Python shell中的交互地试验一下。

习题33 while循环

接下来是一个更在你意料之外的概念——while循环。while循环会一直执行它下面的代码块，直到它对应的布尔表达式为False时才会停下来。

等等，还能跟得上这些术语吧？如果你的某一行是以冒号(:)结尾，那就意味着接下来的内容是一个新的代码块，新的代码块是需要被缩进的。只有将代码用这样的方式格式化，Python才能知道你的目的。如果你不太明白这一点，就回去看看if语句、函数和for循环的章节，直到明白为止。

接下来的习题将训练你的大脑去阅读结构化的代码。这和我们把布尔表达式印到你的大脑中的过程有点类似。

回到while循环，它所做的和if语句类似，也是去检查一个布尔表达式的真假，不一样的是它下面的代码块不是只被执行一次，而是执行完后再调回到while所在的位置，如此重复进行，直到while表达式为False为止。

while循环有一个问题，那就是有时它会永不结束。

为了避免这样的问题，你需要遵循下面的规定。

- 1.尽量少用while循环，大部分时候for循环是更好的选择。
- 2.重复检查你的while语句，确定你测试的布尔表达式最终会变成False。
- 3.如果不确定，就在while循环的结尾打印出你要测试的值。看看它的变化。

在这个习题中，你将通过上面的三件事情学会while循环。

ex33.py

```
1  i = 0
2  numbers = []
3
4  while i < 6:
5      print "At the top i is %d" % i
6      numbers.append(i)
7
8      i = i + 1
9      print "Numbers now: ", numbers
10     print "At the bottom i is %d" % i
11
12
13  print "The numbers: "
14
15  for num in numbers:
16      print num
```


应该看到的结果

习题33 会话

```
$ python ex33.py
At the top i is 0
Numbers now: [0]
At the bottom i is 1
At the top i is 1
Numbers now: [0, 1]
At the bottom i is 2
At the top i is 2
Numbers now: [0, 1, 2]
At the bottom i is 3
At the top i is 3
Numbers now: [0, 1, 2, 3]
At the bottom i is 4
At the top i is 4
Numbers now: [0, 1, 2, 3, 4]
At the bottom i is 5
At the top i is 5
Numbers now: [0, 1, 2, 3, 4, 5]
At the bottom i is 6
The numbers:
0
```

1

2

3

4

5

附加练习

- 1.将这个while循环改成一个函数，将测试条件（ $i < 6$ ）中的6换成一个变量。
 - 2.使用这个函数重写你的脚本，并用不同的数进行测试。
 - 3.为函数添加另外一个参数，这个参数用来定义第8行的+1，这样你就可以让它任意递增了。
 - 4.再使用该函数重写一遍这个脚本，看看效果如何。
 - 5.接下来使用for循环和range把这个脚本再写一遍。还需要中间的递增操作吗？如果不去掉它，会有什么样的结果？
- 很有可能程序跑着停不下来了，这时你只要按着Ctrl再敲C（Ctrl+C），这样程序就会中断下来了。

常见问题回答

for循环和**while**循环有何不同？

for循环只能对一些东西的集合进行循环，**while**循环可以对任何对象进行驯化。然而，**while**循环比起来更难弄对，而一般的任务用**for**循环更容易一些。

循环好难理解啊，我该如何理解？

觉得循环不好理解，很大程度上是因为不会顺着代码的运行方式去理解代码。当循环开始时，它会运行整个代码块，代码块结束后回到开始的循环语句。如果想把整个过程视觉化，可以在循环的各处塞入**print**语句，用来追踪变量的变化过程。你可以在循环之前、循环的第一句、循环中间及循环结尾都放一些**print**语句，研究最后的输出，并试着理解循环的工作过程。

习题34 访问列表的元素

列表的用处很大，但只有能访问里边的内容时它才能发挥出作用来。你已经学会了按顺序读出列表的内容，但如果要得到第 5 个元素该怎么办呢？你需要知道如何访问列表中的元素。访问第一个元素的方法是这样的：

```
animals = ['bear', 'tiger', 'penguin', 'zebra']
```

```
bear = animals[0]
```

你定义一个动物的列表，然后你用0来获取第一个元素？！这是怎么回事啊？因为数学里边就是这样，所以Python的列表也是从0开始的。虽然看上去很奇怪，这样定义其实有它的好处，而且实际上设计成0或者1开头其实都可以，

最好的解释方式是将平时使用数字的方式和程序员使用数字的方式做对比。

假设你在观看上面列表中的四种动物（['bear', 'tiger', 'penguin', 'zebra']）的赛跑，而它们比赛的名次正好跟列表里的次序一样。这是一场很激动人心的比赛，因为这些动物没打算吃掉对方，而且比赛还真的举办起来了。结果你的朋友来晚了，他想知道谁赢了比赛，他会问你“嘿，谁是第0名？”不会的，他会问“嘿，谁是第1名？”

这是因为动物的次序是很重要的。没有第1个就没有第2个，没有第2个也没有第3个。第0个是不存在的，因为0的意思是什么都没有。“什么都没有”怎么赢比赛嘛，完全不合逻辑。这样的数我们称之为“序数”（ordinal number），因为它们表示的是事物的顺序。

而程序员不能用这种方式思考问题，因为他们可以从列表的任何一个位置取出一个元素来。对程序员来说，上述列表更像是一叠卡片。如果他们想要tiger，就抓它出来，如果想要zebra，也一样抓取出来。要随机地抓取列表里的内容，列表的每一个元素都应该有一个地址，或者一个“索引”（index），而最好的方式是使用以0开头的索引。相信我说的这一点吧，这种方式获取元素会更容易。这类数被称为“基数”（cardinal number），它意味着你可以任意抓取元素，所以需要有一个0号元素。

那么，这些知识对于你的列表操作有什么帮助呢？很简单，每次你对自己说“我要第3只动物”时，你需要将“序数”转换成“基数”，只要将前者减1就可以了。第3只动物的索引是2，也就是 penguin。由于你一辈子都在跟序数打交道，所以你需要用这种方式来获得基数，只要减1就都搞定了。

记住：虚数 == 有序，以1开始；基数 == 随机选取，以0开始。

来练习一下。定义一个动物列表，然后跟着做后面的练习，你需要写出所指位置的动物名称。如果我用的是“第1”、“第2”等说法，那说明我用的是序数，所以你需要减去1。如果我给你的是基数（0, 1, 2），你只要直接使用即可。

```
animals = ['bear', 'python', 'peacock', 'kangaroo', 'whale', 'platypus']
```

1.位置1的动物

2.第3只动物

3.第1只动物

4.位置3的动物

5.第5只动物

6.位置3的动物

7.第6只动物

8.位置4的动物

对于上述每一条，以这样的格式写出一个完整的句子：“第1只动物

在位置0，是一只熊”然后倒过来念：“在位置0的是第1只动物，是一只熊”。使用Python检查你的答案。

附加练习

1.上网搜索一下关于序数（ordinal number）和基数（cardinal number）的知识并阅读一下。

2.以你对于这些不同的数字类型的了解，解释一下为什么“January 1, 2010”里是2010而不是2009？（提示：你不能随机挑选年份。）

3.再写一些列表，用一样的方式作出索引，确认自己可以在两种数字之间互相翻译。

4.使用Python检查自己的答案。

警告 会有程序员告诉你让你去阅读一个叫Dijkstra的人写的关于数的话题，我建议你还是不读为妙。除非你喜欢听一个在编程这一行刚兴起时就停止从事编程的人对你大喊大叫。

习题35 分支和函数

你已经学会了if语句、函数，还有列表。现在你要换一换头脑了。
把下面的代码写下来，看你是否能看懂它实现的是什么功能。

ex35.py

```
1  from sys import exit
2
3  def gold_room():
4      print "This room is full of gold.  How much do you take?"
5
6      next = raw_input("> ")
7      if "0" in next or "1" in next:
8          how_much = int(next)
9      else:
10         dead("Man, learn to type a number.")
11
12     if how_much < 50:
13         print "Nice, you're not greedy, you win!"
14         exit(0)
15     else:
16         dead("You greedy bastard!")
17
18
```

```
19 def bear_room():
20     print "There is a bear here."
21     print "The bear has a bunch of honey."
22     print "The fat bear is in front of another door."
23     print "How are you going to move the bear?"
24     bear_moved = False
25
26     while True:
27         next = raw_input("> ")
28
29         if next == "take honey":
30             dead("The bear looks at you then slaps your face
off.")
31         elif next == "taunt bear" and not bear_moved:
32             print "The bear has moved from the door.You can
go through it now."
33             bear_moved = True
34         elif next == "taunt bear" and bear_moved:
35             dead("The bear gets pissed off and chews your leg
off.")
36         elif next == "open door" and bear_moved:
37             gold_room()
38         else:
39             print "I got no idea what that means."
40
41
42 def cthulhu_room():
```

```
43     print "Here you see the great evil Cthulhu."
44     print "He, it, whatever stares at you and you go insane."
45     print "Do you flee for your life or eat your head?"
46
47     next = raw_input("> ")
48
49     if "flee" in next:
50         start()
51     elif "head" in next:
52         dead("Well that was tasty!")
53     else:
54         cthulhu_room()
55
56
57 def dead(why):
58     print why, "Good job!"
59     exit(0)
60
61 def start():
62     print "You are in a dark room."
63     print "There is a door to your right and left."
64     print "Which one do you take?"
65
66     next = raw_input("> ")
67
68     if next == "left":
69         bear_room()
```

```
70     elif next == "right":
71         cthulhu_room()
72     else:
73         dead("You stumble around the room until you starve.")
74
75
76 start()
```

应该看到的结果

下面是我玩游戏的过程。

习题35 会话

```
$ python ex35.py
```

```
You are in a dark room.
```

```
There is a door to your right and left.
```

```
Which one do you take?
```

```
> left
```

```
There is a bear here.
```

```
The bear has a bunch of honey.
```

```
The fat bear is in front of another door.
```

```
How are you going to move the bear?
```

```
> taunt bear
```

```
The bear has moved from the door. You can go through it now.
```

```
> open door
```

```
This room is full of gold. How much do you take?
```

```
> 1000
```

```
You greedy bastard! Good job!
```

附加练习

- 1.把这个游戏的地图画出来，把自己的路线也画出来。
- 2.改正你所有的错误，包括拼写错误。
- 3.为你不懂的函数写注释。记得文档注释该怎么写吗？
- 4.为游戏添加更多元素。通过怎样的方式可以简化并且扩展游戏的功能呢？
- 5.这个gold_room游戏使用了奇怪的方式让你键入一个数。这种方式会导致什么样的bug？你可以用比检查0、1更好的方式判断输入是否是数吗？int()这个函数可以给你一些头绪。

常见问题回答

救命啊！太难了，我搞不懂！

当你搞不懂的时候，就在每一行代码的上方写下注释，向自己解释这一行的功能。在这个过程中如果有了新的理解，就随时修正自己前面的注释。注释完后，就画一个工作原理的示意图，或者写一段文字表述一下。这样你就能弄懂了。

为什么是**while True**？

这样可以创建一个无限循环。

exit(0)有什么功能？

在很多类型的操作系统里，**exit(0)**可以中断某个程序，而其中的数字参数则用来表示程序是否是遇到错误而中断的。**exit(1)**表示发生了错误，而 **exit(0)**则表示程序是正常退出的。这和我们学的布尔逻辑 **0==False**正好相反，不过你可以用不一样的数字表示不同的错误结果。比如，你可以用**exit(100)**来表示另一种和**exit(2)**或**exit(1)**不同的错误。

为什么**raw_input()**有时写成**raw_input('> ')**？

raw_input的参数是一个会被打印出来的字符串，这个字符串一般用来提示用户输入。

习题36 设计和调试

现在你已经学会了if语句，我将给你一些使用for循环和while循环的规则，以免你日后遇到麻烦。我还会教你一些调试的小技巧，以便你能发现自己程序的问题。最后，你将需要设计一个和前一个习题类似的小游戏，不过内容略有更改。

if语句的规则

1.每一条if语句必须包含一个else。

2.如果这个else永远都不应该被执行到，因为它本身没有任何意义，那你必须在else语句后面使用一个叫做die的函数，让它打印出错误信息并且“死”给你看，这和上一个习题类似，这样你可以找到很多的错误。

3.if语句的嵌套不要超过两层，最好尽量保持只有一层。这意味着，如果你在if里边又有了一个if，那你就需要把第二个if移到另一个函数里面。

4.将if语句当做段落来对待，其中的每一个if、elif、else组合就跟一个段落的句子组合一样。在这种组合的最前面和最后面留一个空行以作区分。

5.你的布尔测试应该很简单，如果它们很复杂，你需要将它们的运算事先放到一个变量里，并且为变量取一个好名字。

遵循上面的规则，你就会写出比大部分程序员都好的代码来。回到上一习题中，看看我有没有遵循这些规则，如果没有的话，就将其改正过来。

警告 在日常编程中不要成为这些规则的奴隶。在训练中，你需要通过这些规则的应用来巩固学到的知识，而在实际编程中这些规则有时其实很蠢。如果你觉得哪个规则很蠢，就别使用它。

循环的规则

1.只有在循环永不停止时使用“while 循环”，这意味着你可能永远都用不到。这条只有Python中成立，其他的语言另当别论。

2.其他类型的循环都使用for循环，尤其是在循环的对象数量固定或者有限的情况下。

调试的小技巧

- 1.不要使用“调试器”（debugger）。调试器所做的相当于对病人的全身扫描。你并不会得到某方面的有用信息，而且你会发现它输出的信息太多，而且大部分没有用，或者只会让你更困惑。
- 2.最好的调试程序的方法是使用 `print` 在各个想要检查的关键环节将关键变量打印出来，从而检查那里是否有错。
- 3.让程序一部分一部分地运行起来。不要等一个很长的脚本写完后才去运行它，写一点，运行一点，再修改一点。

家庭作业

写一个和上一个习题类似的游戏。同类的任何题材的游戏都可以，花一个星期让它尽可能有趣一些。作为附加练习，你可以尽量多使用列表、函数及模块（记得习题13吗？），而且尽量多弄一些新的Python代码让你的游戏跑起来。

不过有一点需要注意，你应该把游戏的设计先写出来。在你写代码之前，你应该设计出游戏的地图，创建出玩家会遇到的房间、怪物及陷阱等环节。

一旦搞定了地图，就可以写代码了。如果你发现地图有问题，就调整一下地图，让代码和地图互相符合。

最后一个建议：每一个程序员在开始一个新的大项目时都会被非理性的恐惧影响到，为了避免这种恐惧，他们会拖延时间，到最后一事无成。我有时就会这样，每个人都会有这样的经历，避免这种情况的最好方法是把自己要做的事情列出来，一次完成一样。

开始做吧。先做一个小一点的版本，扩充它，让它变大，把自己要完成的事情一一列出来，然后逐个完成就可以了。

习题37 复习各种符号

现在该复习学过的符号和Python关键字了，而且你在这个习题中还会学到一些新的东西。我在这里所做的是将所有的Python符号和关键字列出来，这些都是值得掌握的重点。

在这个习题中，你需要复习每一个关键字，从记忆中想起它的作用并且写下来，接着上网搜索它真正的功能。有些内容可能是无法搜的，所以这对你可能有些难度，不过你还是需要坚持尝试。

如果你发现记忆中的内容有误，就在索引卡片上写下正确的定义，试着将自己的记忆纠正过来。如果你就是不知道它的定义，就把它直接写下来，以后再做研究。

最后，将每一种符号和关键字用在程序里，你可以用一个小程序来做，也可以尽量多写一些程序来巩固记忆。这里的关键点是明白各个符号的作用，确认自己没搞错，如果搞错了就纠正过来，然后将其用在程序里，并且通过这样的方式加深自己的记忆。

关键字

and

del

from

not

while

as

elif

global

or

with

assert

else

if

pass

yield

break

except

import

print

class

exec

in

raise
continue
finally
is
return
def
for
lambda
try

数据类型

针对每一种数据类型，都举出一些例子来，例如，针对string，你可以举出一些字符串，针对number，你可以举出一些数字。

True

False

None

strings

numbers

floats

lists

字符串转义序列

对于字符串“转义序列”（escape sequence），需要在字符串中应用它们，确认自己清楚地知道它们的功能。

\\

\'

\"

\a

\b

\f

\n

\r

\t

\v

字符串格式化

一样的，在字符串中使用它们，确认它们的功能。

`%d`

`%i`

`%o`

`%u`

`%x`

`%X`

`%e`

`%E`

`%f`

`%F`

`%g`

`%G`

`%c`

`%r`

`%s`

`%%`

操作符

有些操作符你可能还不熟悉，还是一一看一下，研究一下它们的功能，如果研究不出来也没关系，记录下来日后解决。

+

-

*

**

/

//

%

<

>

<=

>=

==

!=

<>

()

[]

{}

@

,

:

·

=

;

+=

--

*=

/=

//=

%=

**=

花一个星期学习这些东西，如果能提前完成就更好了。我们的目的是覆盖所有的符号类型，确认你已经牢牢记住它们。另外很重要的一点是，这样你可以找出自己还不知道哪些东西，为日后学习找到一些方向。

阅读代码

现在去找一些Python代码阅读一下。你需要自己找代码，然后从中学习一些东西。你学到的东西已经足够让你看懂一些代码了，但你可能还无法理解这些代码的功能。这个联系我会教你如何运用学到的东西理解别人的代码。

首先把你想要理解的代码打印到纸上。没错，你需要打印出来，因为和屏幕输出相比，你的眼睛和大脑更习惯于接受纸质打印的内容。一次最多打印几页就可以了。

然后通读打印出来的代码并做好标记，标记的内容包括以下几个方面。

- 1.函数以及函数的功能。
- 2.每个变量的初始赋值。
- 3.每个在程序的各个部分中多次出现的变量。它们以后可能会给你带来麻烦。
- 4.任何不包含else的if语句。它们是正确的吗？
- 5.任何可能没有结束点的while循环。
- 6.最后一条，代码中任何你看不懂的部分都记下来。

接下来你需要通过注释的方式向自己解释代码的含义。解释各个函数的使用方法，各个变量的用途，以及任何其他方面的内容，只要能帮助你理解代码即可。

最后，在代码中比较难的各个部分，逐行或者逐个函数跟踪变量值。你可以再打印一份出来，在空白处写出要“追踪”的每个变量的值。

一旦基本理解了代码的功能，回到电脑面前，在屏幕上重读一次，

看看能不能找到新的问题点。然后继续找新的代码，用上述的方法去阅读理解，直到你不再需要纸质打印为止。

附加练习

- 1.研究一下什么是“流程图”（flow chart），并学着画一下。
- 2.如果在读代码的时候找出了错误，试着把它们改对，并把修改内容发给作者。
- 3.不使用纸质打印时，可以使用注释符号#在程序中加入笔记。有时这些笔记会对后来的读代码的人有很大的帮助。

常见问题回答

%d和**%i**有何不同？

没有什么不同，只不过由于历史原因，用%d的人更多一些而已。

怎样在网上搜索这些东西？

在要搜索的东西前面加上“python”就可以了，比如说你要搜索yield，就输入python yield。

习题38 列表的操作

你已经学过了列表。在学习while循环的时候，你对列表进行过“追加”（append）操作，而且将列表的内容打印了出来。另外你应该还在附加练习里研究过Python文档，看了列表支持的其他操作。这已经是一段以前了，所以如果你不记得了的话，就回到本书的前面再复习一遍把。

找到了吗？还记得吗？很好。那时候你对一个列表执行了append函数。不过，你也许还没有真正明白发生的事情，所以我们再来看看可以对列表进行什么样的操作。

当你看到mystuff.append('hello')这样的代码时，事实上已经在Python内部激发了一个连锁反应。以下是它的工作原理。

1. Python 看到你用到了 mystuff，于是就去找到这个变量。也许它需要倒着检查你有没有在哪里用=创建过这个变量，或者检查它是不是一个函数参数，或者看它是不是一个全局变量。不管哪种方式，它得先找到mystuff这个变量才行。

2. 一旦它找到了mystuff，就轮到处理句点（.）这个操作符，而且开始查看mystuff内部的一些变量了。由于mystuff是一个列表，Python知道mystuff支持一些函数。

3. 接下来轮到了处理append。Python会将append和mystuff支持的所有函数的名称一一对比，如果确实其中有一个叫 append 的函数，那么Python 就会去使用这个函数。

4. 接下来Python看到了括号（()）并意识到：“噢，原来这应该是一

个函数。”到了这里，它就正常会调用这个函数了，不过这里的函数还要多一个参数才行。

5.这个额外的参数其实是.....mystuff！我知道，很奇怪是不是？不过这就是Python的工作原理，所以还是记住这一点，就当它是正常的好了。真正发生的事情其实是append(mystuff, 'hello')，不过你看到的只是mystuff.Append('hello')。

大部分时候你不需要知道这些细节，不过如果你看到一个像这样的Python 错误信息的时候，上面的细节就对你有用了：

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> class Thing(object):
...     def test(hi):
...         print "hi"
...
>>> a = Thing()
>>> a.test("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: test() takes exactly 1 argument (2 given)
>>>
```

就是这个吗？嗯，这个是我在 Python 命令行下展示给你的一点“魔法”。你还没有见过class，不过后面很快就要见到了。现在你看到Python 说 test() takes exactly 1 argument (2 given)（test()只可以接收两个参数，实际上给了一个）。这意味着，Python把a.test("hello")改成了test(a, "hello")，而有人弄错了，没有为它添加a这个参数。

一下子要消化这么多可能有点难度，不过下面会做几个练习，让你头脑中有一个深刻的印象。下面的习题将字符串和列表混在一起，看看你能不能在里边找出点乐子来。

ex38.py

```
1  ten_things = "Apples Oranges Crows Telephone Light Sugar"
2
3  print "Wait there's not 10 things in that list, let's fix that."
4
5  stuff = ten_things.split(' ')
6  more_stuff = ["Day", "Night", "Song", "Frisbee", "Corn", "Banana",
"Girl", "Boy"]
7
8  while len(stuff) != 10:
9      next_one = more_stuff.pop()
10     print "Adding: ", next_one
11     stuff.append(next_one)
12     print "There's %d items now." % len(stuff)
13
14  print "There we go: ", stuff
15
16  print "Let's do some things with stuff."
17
18  print stuff[1]
19  print stuff[-1] # whoa! fancy
20  print stuff.pop()
21  print ''.join(stuff) # what? cool!
22  print '#'.join(stuff[3:5]) # super stellar!
```

应该看到的结果

习题38 会话

```
$ python ex38.py
Wait there's not 10 things in that list, let's fix that.
Adding: Boy
There's 7 items now.
Adding: Girl
There's 8 items now.
Adding: Banana
There's 9 items now.
Adding: Corn
There's 10 items now.
There we go: ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light', 'Sugar',
'Boy'
              , 'Girl', 'Banana', 'Corn']
Let's do some things with stuff.
Oranges
Corn
Corn
Apples Oranges Crows Telephone Light Sugar Boy Girl Banana
Telephone#Light
```

附加练习

1.将每一个被调用的函数以上述的方式翻译成 Python 实际执行的动作。例如，`''.join(things)`其实是`join('', things)`。

2.将这两种方式翻译为自然语言。例如，`''.join(things)`可以翻译成“用''连接things”，而`join('', things)`的意思是“为''和things调用join函数”。这其实是同一件事情。

3.上网阅读一些关于“面向对象编程”（Object Oriented Programming, OOP）的资料。

晕了吧？嗯，我以前也是。别担心。你将从这本书学到足够的关于面向对象编程的基础知识，以后你还可以慢慢学到更多。

4.查一下Python中的“类”（class）是什么东西。不要阅读关于其他语言的“类”的用法，这会让你更糊涂。

5.`dir(something)`和something的class有什么关系？

6.如果你不知道我讲的是些什么东西，别担心。程序员为了显得自己聪明，发明了“面向对象编程”，然后他们就开始滥用这个东西了。如果你觉得这东西太难，可以开始学一下“函数式编程”（functional programming）。

常见问题回答

你不是说别用**while**循环吗？

是的。要记住，有时候如果你有很好的理由，那么规则也是可以打破的。死守着规则不放时不明智的。

stuff[3:5]实现了什么功能？

这是一个列表“切片”动作，它会从stuff列表的第3个元素开始取值，直到第5个元素。注意，这里并不包含第5个元素，这跟range(3,5)的情况是一样的。

为什么**join(' ', stuff)**不灵？

join的文档写得有问题。其实它不是这么工作的，它是你要插入的字符串的一个方法函数，函数的参数是你连接的字符串构成的数组，所以应该写作'**stuff.join(' ')**'。

习题39 字典，可爱的字典

接下来我要教你另外一种让你伤脑筋的容器，一旦学会这种容器，你将拥有超酷的能力。这是最有用的容器——字典（**dictionary**）。

Python将其称为“字典”，有的语言里它的名称是“散列”。这两种名字我都会用到，不过这并不重要，重要的是它们和列表的区别。你看，针对列表你可以做这样的事情：

```
>>> things = ['a', 'b', 'c', 'd']
```

```
>>> print things[1]
```

```
b
```

```
>>> things[1] = 'z'
```

```
>>> print things[1]
```

```
z
```

```
>>> print things
```

```
['a', 'z', 'c', 'd']
```

```
>>>
```

你可以使用数字作为列表的索引，也就是可以通过数找到列表中的元素。到目前为止，你应该了解这一点，但是请确定你已经理解：你只能使用数字来获取列表中的项。

而字典所做的是，让你可以通过任何东西（不只是数字）找到元素。是的，字典可以将一个物件和另外一个东西关联，不管它们的类型是什么，我们来看看：

```
>>> stuff = {'name': 'Zed', 'age': 36, 'height': 6*12+2}
```

```
>>> print stuff['name']
Zed
>>> print stuff['age']
36
>>> print stuff['height']
74
>>> stuff['city'] = "San Francisco"
>>> print stuff['city']
San Francisco
>>>
```

你将看到除了通过数字以外，还可以用字符串来从字典中获取stuff，我们还可以用字符串来往字典中添加元素。当然它支持的不只有字符串，我们还可以做这样的事情：

```
>>> stuff[1] = "Wow"
>>> stuff[2] = "Neato"
>>> print stuff[1]
Wow
>>> print stuff[2]
Neato
>>> print stuff
{'city': 'San Francisco', 2: 'Neato',
 'name': 'Zed', 1: 'Wow', 'age': 36,
 'height': 74}
>>>
```

在这里我使用了两个数字。其实我可以使用任何东西，不过这么说并不准确，你先这么理解就行了。

当然了，一个只能放东西进去的字典是没啥意思的，所以我们还要

有删除物件的方法，也就是使用del这个关键字：

```
>>> del stuff['city']
>>> del stuff[1]
>>> del stuff[2]
>>> stuff
{'name': 'Zed', 'age': 36, 'height': 74}
>>>
```

接下来要做一个练习，你必须非常仔细，我要求你将这个习题写下来，然后试着弄懂它做了些什么。这个习题很有趣，做完以后你可能会豁然开朗的感觉。

ex39.py

```
1  # create a mapping of state to abbreviation
2  states = [
3      'Oregon': 'OR',
4      'Florida': 'FL',
5      'California': 'CA',
6      'New York': 'NY',
7      'Michigan': 'MI'
8  ]
9
10 # create a basic set of states and some cities in them
11 cities = [
12     'CA': 'San Francisco',
13     'MI': 'Detroit',
14     'FL': 'Jacksonville'
15 ]
16
```

```
17 # add some more cities
18 cities['NY'] = 'New York'
19 cities['OR'] = 'Portland'
20
21 # print out some cities
22 print '-' * 10
23 print "NY State has: ", cities['NY']
24 print "OR State has: ", cities['OR']
25
26 # print some states
27 print '-' * 10
28 print "Michigan's abbreviation is: ", states['Michigan']
29 print "Florida's abbreviation is: ", states['Florida']
30
31 # do it by using the state then cities dict
32 print '-' * 10
33 print "Michigan has: ", cities[states['Michigan']]
34 print "Florida has: ", cities[states['Florida']]
35
36 # print every state abbreviation
37 print '-' * 10
38 for state, abbrev in states.items():
39     print "%s is abbreviated %s" % (state, abbrev)
40
41 # print every city in state
42 print '-' * 10
43 for abbrev, city in cities.items():
```

```
44     print "%s has the city %s" % (abbrev, city)
45
46 # now do both at the same time
47 print '-' * 10
48 for state, abbrev in states.items():
49     print "%s state is abbreviated %s and has city %s" % (
50         state, abbrev, cities[abbrev])
51
52 print '-' * 10
53 # safely get a abbreviation by state that might not be there
54 state = states.get('Texas', None)
55
56 if not state:
57     print "Sorry, no Texas."
58
59 # get a city with a default value
60 city = cities.get('TX', 'Does Not Exist')
61 print "The city for the state 'TX' is: %s" % city
```

应该看到的结果

习题39 会话

```
$ python ex39.py
```

```
-----
```

```
NY State has: New York
```

```
OR State has: Portland
```

```
-----
```

```
Michigan's abbreviation is: MI
```

```
Florida's abbreviation is: FL
```

```
-----
```

```
Michigan has: Detroit
```

```
Florida has: Jacksonville
```

```
-----
```

```
California is abbreviated CA
```

```
Michigan is abbreviated MI
```

```
New York is abbreviated NY
```

```
Florida is abbreviated FL
```

```
Oregon is abbreviated OR
```

```
-----
```

```
FL has the city Jacksonville
```

```
CA has the city San Francisco
```

```
MI has the city Detroit
```

```
OR has the city Portland
```

NY has the city New York

California state is abbreviated CA and has city San Francisco

Michigan state is abbreviated MI and has city Detroit

New York state is abbreviated NY and has city New York

Florida state is abbreviated FL and has city Jacksonville

Oregon state is abbreviated OR and has city Portland

Sorry, no Texas.

The city for the state 'TX' is: Does Not Exist

附加练习

1.在Python文档中找到字典（`dictionary`，又称作`dicts`或`dict`）的相关内容，学着对字典做更多的操作。

2.找出一些字典无法做到的事情。例如，比较重要的一个就是字典的内容是无序的，你可以检查一下看看是否真是这样。

3.试着把`for`循环执行到字典上面，然后试着在`for`循环中使用字典的`items()`函数，看看会有什么样的结果。

常见问题回答

列表和字典有何不同？

列表是有序排列的一些物件，而字典是将一些物件（键）对应到另外一些物件（值）的数据结构。

字典能用在哪里？

各种需要通过某个值去查看另一个值的场合。事实上，可以把字典当做“查询表”。

列表能用在哪里？

列表是专供有序排列的数据使用的。只要知道索引就能查到对应的值了。

有没有办法弄一个可以排序的字典？

看看Python里的collections.OrderedDict数据结构。上网搜索一下文档和用法。

习题40 模块、类和对象

Python是一种面向对象编程（**OOP**）语言。这个说法的意思是，**Python**里边有一种叫做类（**class**）的结构，通过它可以用一种特殊的方式构造软件。通过使用类，可以让程序架构更为整齐，使用起来也会更为干净——至少理论上应该是这样的。

现在我要教你的是面向对象编程的初步知识，我会用你学过的知识介绍面向对象编程、类及对象。问题是面向对象编程本身就是个奇怪的东西，只有努力去弄懂这个习题的内容，好好写代码，然后到下一个习题就能把**OOP**像钉钉子一样钉到脑子里了。

现在就开始吧。

模块和字典差不多

你知道怎样创建和使用字典，这是一种将一种东西对应到另外一种的方式。这意味着，如果你有一个字典，它里边有一个叫'apple'的键，而你要从中取值的话，你需要这样做：

```
mystuff = {'apple': "I AM APPLES!"}
print mystuff['apple']
```

记住这个“从Y获取X”的概念，现在再来看看模块（module），你已经创建和使用过一些模块了，已经了解了它们的一些属性。

- 1.模块是包含函数和变量的Python文件。
- 2.可以导入这个文件。
- 3.然后可以使用.操作符访问模块中的函数和变量。

假如说我有一个模块名字叫mystuff.py，并且里边放了个叫做apple的函数，就像这样：

```
# this goes in mystuff.py
def apple():
    print "I AM APPLES!"
```

接下来就可以用import来调用这个模块，并且访问apple函数：

```
import mystuff
mystuff.apple()
```

还可以放一个叫tangerine的变量到模块里边：

```
def apple():
    print "I AM APPLES!"
# this is just a variable
```

```
tangerine = "Living reflection of a dream"
```

同样，还是可以访问这个变量：

```
import mystuff
```

```
mystuff.apple()
```

```
print mystuff.tangerine
```

回到字典的概念，你会发现这和字典的使用方式有点儿相似，只不过语法不同而已。我们来比一比：

```
mystuff['apple'] # get apple from dict
```

```
mystuff.apple() # get apple from the module
```

```
mystuff.tangerine # same thing, it's just a variable
```

也就是说，Python里边有这么一个通用的模式：

1. 拿一个类似“键=值”风格的容器；
2. 通过“键”的名称获取其中的“值”。

对于字典来说，键是一个字符串，获得值的语法是“[键]”。对于模块来说，键是函数或者变量的名称，而语法是“.”。除了这个，它们基本上就没什么区别了。

类和模块差不多

模块还可以用一种方法去理解：可以把它们当做一种特殊的字典，通过它们可以存储一些Python 代码，可以通过“.”操作符访问这些代码。Python 还有另外一种代码结构用来实现类似的目的，那就是类，通过类，你可以把一组函数和数据放到一个容器中，从而用“.”操作符访问它们。

如果要用创建mystuff模块的方法来创建一个类，那么方法大致是这样的：

```
class MyStuff(object):
    def __init__(self):
        self.tangerine = "And now a thousand years between"
    def apple(self):
        print "I AM CLASSY APPLES!"
```

这个和模块比起来有些复杂，确实，与模块相比，这里的确做了很多事情。不过你应该能大致看出来，这段代码差不多就是模拟了一个名字叫 MyStuff 的迷你模块，里边有一个叫apple()的函数，难懂的恐怕是__init__()函数，还有就是设置 tangerine 变量时用了self.tangerine这样的语法。

使用类而非模块的原因如下：可以拿上面这个类重复创建出很多出来，哪怕是一次一百万个，它们也会互不干涉。而对于模块来说，一次导入之后，整个程序里就只有这么一份内容，只有鼓捣得很深才能弄点花样出来。

不过在弄懂这个之前，你要先理解“对象”（object）是什么东西，

以及如何使用MyStuff达到类似mystuff.py模块的结果。

对象相当于迷你导入

如果说类和迷你模块差不多，那么对于类来说，也必然有一个类似导入（import）的概念。这个概念名称就是“实例化”（instantiate）。这只是一中故作高深的叫法而已，它的意思其实是“创建”。当你将一个类“实例化”以后，就得到了一个对象（object）。

实现实例化的方法就是像调用函数一样地调用一个类：

```
thing = MyStuff()
thing.apple()
print thing.tangerine
```

第一行代码就是“实例化”操作，这和调用函数很相似。然而，当你进行实例化操作时，Python在背后做了一系列的工作，下面就针对上面的代码详细解释一下。

- 1.Python看到了MyStuff()并且知道了它是你定义过的一个类。
- 2.Python创建了一个空对象，里边包含了你在该类中用def创建的所有函数。
- 3.然后Python回去检查你是不是在里边创建了一个__init__函数，如果有创建，它就会调用这个函数，从而对你新创建的空对象实现了初始化。
- 4.在MyStuff的__init__函数里，有一个多余的函数叫做self，这就是Python为我们创建的空对象，而我可以对它进行类似模块、字典等的操作，为它设置一些变量进去。
- 5.在这里，我把self.tangerine设成了一段歌词，这样我就初始化了该对象。

6.最后Python将这个新建的对象赋给一个叫thing的变量，以供后面使用。

这就是当你像调用函数一样调用类的时候，Python 完成这个“迷你导入”的过程。记住这不是拿来一个类就直接用，而是将类当做一个“蓝图”，然后用它创建和这个类有相同属性的副本。

提醒一点，我的解释和Python的实际原理还是有一点小小的出入的，在这里，基于你现有的关于模块的知识，我也只能暂时这么解释了。事实上，类和对象与模块是完全不同的东西。如果实实在在地跟你讲，我大概会说出下面这些内容。

类就像一种蓝图或者一种预定义的东西，通过它可以创建新的迷你模块。

实例化的过程相当于你创建了这么一个迷你模块，而且同时导入了它。

结果创建的迷你模块就是一个对象，你可以将它赋予一个变量并进行后续操作。

通过这一系列的操作，类和对象与模块已经很不同了，所以这里的内容只是为了帮你理解类的概念而已。

获取某样东西里包含的东西

现在有三种方法可以从某个东西里获取东西：

dict style

mystuff['apples']

module style

mystuff.apples()

print mystuff.tangerine

class style

thing = MyStuff()

thing.apples()

print thing.tangerine

第一个关于类的例子

你应该注意到了这三种“键=值”容器类型的相似性，而且有一些问题要问。先别问，下一个习题会让你了解面向对象编程的一些专有词汇。在这个习题里，我只要求你写代码并让它运行起来，有了经验才能继续前进。

ex40.py

```
1  class Song(object):
2
3      def __init__(self, lyrics):
4          self.lyrics = lyrics
5
6      def sing_me_a_song(self):
7          for line in self.lyrics:
8              print line
9
10 happy_bday = Song(["Happy birthday to you",
11                    "I don't want to get sued",
12                    "So I'll stop right there"])
13
14 bulls_on_parade = Song(["They rally around the family",
15                          "With pockets full of shells"])
16
17 happy_bday.sing_me_a_song()
```


18

19 `bulls_on_parade.sing_me_a_song()`

应该看到的结果

习题40 会话

```
$ python ex40.py
```

```
Happy birthday to you
```

```
I don't want to get sued
```

```
So I'll stop right there
```

```
They rally around the family With pockets full of shells
```

附加练习

1.使用这种方式写更多的歌进去，确定自己弄懂了传入的歌词是一个字符串列表。

2.将歌词放到另一个变量里，然后再类里使用这个新定义的变量。

3.试着看能不能给它加些新功能，不知道怎么做也没关系，只要试着去做就行，弄坏了也没关系。

4.在网上搜索一下“object oriented programming”（面向对象编程），给自己洗洗脑。弄不懂也没关系，其实里边有一半的东西对我来说也是没有意义的。

常见问题回答

为什么创建__init__或者别的类函数时需要多加一个self变量？

如果不加self，cheese = 'Frank'这样的代码意义就不明确了，它指的既可能是实例的cheese属性，也可能是一个叫做cheese的局部变量。有了self.cheese = 'Frank'就清楚地知道这指的是实例的属性self.cheese。

习题41 学习面向对象术语

在这个习题中我将教你面向对象的术语。我会给你一些你需要知道的词汇的定义，然后给你一系列需要你填空的句子，你要按自己的理解将其补充完整，最后我会给你很多练习，以巩固这些新词汇。

单词练习

类（**class**）：告诉Python创建新类型的东西。

对象（**object**）：两个意思，即最基本的东西，或者某个东西的实例。

实例（**instance**）：这是让Python创建一个时得到的东西。

def：这是在类里边定义函数的方法。

self：在类的函数中，**self**指代被访问的对象或者实例的一个变量。

继承（**inheritance**）：指一个类可以继承另一个类的特性，和父子关系类似。

组合（**composition**）：指一个类可以将别的类作为它的部件构建起来，有点儿像车子和车轮的关系。

属性（**attribute**）：类的一个属性，它来自于组合，而且通常是一个变量。

是什么（**is-a**）：用来描述继承关系，如Salmon is-a Fish（鲑鱼是一种鱼）。

有什么（**has-a**）：用来描述某个东西是由另外一些东西组成的，或者某个东西有某个特征，如Salmon has-a mouth（鲑鱼有一张嘴）。

好了，花时间做几张速记卡，把它们记下来。一样的，一开始看上去这些东西没什么意义，但是为了做这个习题，需要先把它们记下来，后面会慢慢理解这些术语的。

语汇练习

接下来我给出了一些代码，以及用来描述代码的句子。

`class X(Y):` 创建一个叫X的类，它是Y的一种。

`class X(object): def __init__(self, J):` 类X有一个__init__接收self和J作为参数。

`class X(object): def M(self, J):` 类X有一个函数名称为M，它接收self和J作为参数。

`foo = X():` 将foo设为类X的一个实例。

`foo.M(J):` 从foo中找到M函数，并使用self和J参数调用它。

`foo.K = Q:` 从foo中获取K属性，并将其设为Q。

每一条当中，你看到X、Y、M、J、K、Q及foo的地方，都可以将它们当作空白点来对待。例如，还可以将句子写成下面这样。

1. 创建一个叫???的类，它是Y的一种。

2. 类???有一个__init__，能接收self和???作为参数。

3. 类???有一个函数名称为???，可以接收self和???作为参数。

4. 将foo设为class ???的一个实例。

5. 从foo中找到???函数，并使用self和???参数调用它。

6. 从foo中获取???属性，并将其设为???。

一样的，将这些写在速记卡上并记下来。将Python代码放在正面，解释的句子放在背面。每次看到同样语法格式的代码，你都应该能准确地说出它的含义。大致相同是不够的，要做到完全相同。

混合巩固练习

最后给你的准备材料是将单词练习和语汇练习搭配起来。

- 1.拿一张短语卡并且记下来。
- 2.翻到背面阅读句子，然后针对句子中的每个专有名词，找到对应的单词卡片。
- 3.再去记忆这些句子的单词。
- 4.持续练习，直到烦了为止，然后可以休息一下接着记。

阅读测试

现在有一小段Python代码，利用这段代码你可以无穷尽地去记忆这里的专有词汇。这段代码很简单，你应该能看明白，它唯一的功能就是调用了一个叫urllib的库，然后下载这些专有词汇。你应该把下面这段脚本输入并保存到oop_test.py代码文件中，然后运行它。

ex41.py

```
1  import random
2  from urllib import urlopen
3  import sys
4
5  WORD_URL = "http://learncodethehardway.org/words.txt"
6  WORDS = []
7
8  PHRASES = {
9      "class %%%(%%%)":
10         "Make a class named %%% that is-a %%%.",
11         "class %%%(object):\n\tdef __init__(self, ***)" :
12         "class %%% has-a __init__ that takes self and ***
parameters.",
13         "class %%%(object):\n\tdef ***(self, @@@)":
14         "class %%% has-a function named *** that takes self and
@@@ parameters.",
15         "**** = %%%()":
```

```

16         "Set *** to an instance of class %%%.",
17         "****.***(@@@)":
18         "From *** get the *** function, and call it with parameters
self, @@@.",
19         "****.*** = '****'":
20         "From *** get the *** attribute and set it to '***'."
21     }
22
23     # do they want to drill phrases first
24     PHRASE_FIRST = False
25     if len(sys.argv) == 2 and sys.argv[1] == "english":
26         PHRASE_FIRST = True
27
28     # load up the words from the website
29     for word in urlopen(WORD_URL).readlines():
30         WORDS.append(word.strip())
31
32
33     def convert(snippet, phrase):
34         class_names = [w.capitalize() for w in
35             random.sample(WORDS,
snippet.count("%%%"))]
36         other_names = random.sample(WORDS,
snippet.count("****"))
37         results = []
38         param_names = []
39

```

```
40         for i in range(0, snippet.count("@@@")):
41             param_count = random.randint(1,3)
42             param_names.append(', '.join(random.sample(WORDS,
param_count))))
43
44         for sentence in snippet, phrase:
45             result = sentence[:]
46
47             # fake class names
48             for word in class_names:
49                 result = result.replace("%%%", word, 1)
50
51             # fake other names
52             for word in other_names:
53                 result = result.replace("***", word, 1)
54
55             # fake parameter lists
56             for word in param_names:
57                 result = result.replace("@@@", word, 1)
58
59             results.append(result)
60
61         return results
62
63
64 # keep going until they hit CTRL-D
65 try:
```

```
66     while True:
67         snippets = PHRASES.keys()
68         random.shuffle(snippets)
69
70         for snippet in snippets:
71             phrase = PHRASES[snippet]
72             question, answer = convert(snippet, phrase)
73             if PHRASE_FIRST:
74                 question, answer = answer, question
75
76             print question
77
78             raw_input("> ")
79             print "ANSWER:  %s\n\n" % answer
80 except EOFError:
81     print "\nBye"
```

运行这段脚本，并将面向对象术语翻译成普通话。你应该能看到 PHRASES 字典结构中有两种不同格式，只要输入正确的就可以了。

练习从语言到代码

接下来你应该用`english`参数运行脚本，这样就可以反向练习了。

记住，这些短语用了一些毫无意义的单词。学习阅读代码的一部分就是读到变量名时不要过多地想象它的意义。很多时候人看到某个词，会走神，比如看到“Cork”，因为他不确定它究竟是什么意思。上面的例子中，“Cork”只是一个任选的变量名称而已。不要过多想它的意思，好好做练习就可以了。

[阅读更多代码](#)

现在你要去找更多的代码，用上面习题中用到的语汇来阅读。你要找到所有带类的文件，然后按下面步骤完成。

- 1.针对每一个类，指出它的名称，以及它是继承于哪个类的。
- 2.列出每个类中的所有函数，以及这些函数的参数。
- 3.列出类中用在self上的所有属性。
- 4.针对每一个属性，指出它是来自哪个类。

这样做的目的是让你学着将学到的短语和它们的实际应用匹配起来。如果你记的足够深刻，就应该可以灵活套用各种短语到实际代码中了。

常见问题回答

result = sentence[:]是什么意思？

这是Python中复制列表的方法。你正在使用“列表切片”（list slicing）语法[:]对列表中的所有元素进行切片操作。

这段脚本好难运行！

到现在为止，你应该有能力输入代码并让它运行起来了。里边确实有一些小的迷惑人的地方，不过没有什么复杂的东西。就用你学到的知识调试脚本就行了。

习题42 对象、类及从属关系

有一个重要的概念需要弄明白，那就是“类”（class）和“对象”（object）的区别。问题在于，类和对象并没有真正的不同。它们其实是同样的东西，只是在不同的时间名字不同罢了。我用禅语来解释一下吧：

鱼和泥鳅有什么区别？

这个问题有没有让你有点儿晕呢？说真的，坐下来想一分钟。我的意思是说，鱼和泥鳅是不一样，不过它们其实也是一样的，是不是？泥鳅是鱼的一种，所以说没什么不同，不过泥鳅又有些特别，它和别的种类的鱼的确不一样，比如泥鳅和黄鳝就不一样，所以泥鳅和鱼既相同又不同。怪了。

这个问题让人晕的原因是大部分人不会这样去思考问题，其实每个人都懂这一点，你无须去思考鱼和泥鳅的区别，因为你知道它们之间的关系。你知道泥鳅是鱼的一种，而且鱼还有别的种类，根本就没必要去思考这类问题。

让我们更进一步，假设你有一只水桶，里边有三条泥鳅。假设你的好人卡多到没地方用，于是你给它们分别取名叫小方、小乔、小丽。现在想想这个问题：

小丽和泥鳅有什么区别？

这个问题一样的奇怪，但比起鱼和泥鳅的问题来还好点。你知道小丽是一条泥鳅，所以他并没有什么不同，他只是泥鳅的一个“实例”（instance）。小斌和小星一样也是泥鳅的实例。我说的“实例”是指

什么呢？我的意思是说它们是从泥鳅创建出来，而且具有泥鳅属性的具体、真实的东西。

所以我们的思维方式是（你可能会有点儿不习惯）：鱼是一个“类”，泥鳅是一个“类”，而小丽是一个“对象”。仔细想想，然后我再一点一点慢慢给你解释。

鱼是一个“类”，表示它不是一个具体的东西，而是一个用来描述具有同类属性的实例的概括性的词汇。有鳍？有鳃？住在水里？好吧，那它就是鱼。

后来水产博士路过，看到你的水桶，于是告诉你：“小伙子，这是鲤形目鳅科的泥鳅。”专家一出，真相即现，并且专家还定义了一个新的叫做“泥鳅”的“类”，而这个“类”又有它特定的属性。细长条？有胡须？爱钻泥巴？炖着吃味道还可以？那它就是一条泥鳅。

最后大厨路过，他跟水产博士说：“什么乱七八糟的，你看那条泥鳅，我叫它小丽，我要把小丽和剁椒配一起做一道小菜。”于是你就有了一只叫做小丽的泥鳅的“实例”（泥鳅也是鱼的一个“实例”），并且你使用了它（把它塞到你的胃里了），这样它就是一个“对象”。

这回你应该了解了：小丽是泥鳅的成员，而泥鳅又是鱼的成员。这里的关式：对象属于某个类，而某个类又属于另一个类。

代码写成什么样子

这个概念有点绕人，不过实话说，你只要在创建和使用类的时候操心一下就可以了。我来给你两个区分类和对象的小技巧。

首先针对类和对象，你需要学会两个说法，“is-a”（是什么）和“has-a”（有什么）。“是什么”要用在谈论“两者以类的关系互相关联”的时候，而“有什么”要用在“两者无共同点，仅是互为参照”的时候。

接下来，通读这段代码，将每一个注释为##??的位置标明它是“is-a”还是“has-a”的关系，并讲明白这个关系是什么。在代码的开始我还举了几个例子，所以你只要写剩下的就可以了。

记住，“是什么”指的是鱼和泥鳅的关系，而“有什么”指的是泥鳅和鳃的关系[\[1\]](#)。

ex42.py

```
1  ## Animal is-a object (yes, sort of confusing) look at the extra credit
2  class Animal(object):
3      pass
4
5  ## ??
6  class Dog(Animal):
7
8      def __init__(self, name):
9          ## ??
10         self.name = name
```

```
11
12  ## ??
13  class Cat(Animal):
14
15      def __init__(self, name):
16          ## ??
17          self.name = name
18
19  ## ??
20  class Person(object):
21
22      def __init__(self, name):
23          ## ??
24          self.name = name
25
26          ## Person has-a pet of some kind
27          self.pet = None
28
29  ## ??
30  class Employee(Person):
31
32      def __init__(self, name, salary):
33          ## ?? hmm what is this strange magic?
34          super(Employee, self).__init__(name)
35          ## ??
36          self.salary = salary
37
```

```
38  ## ??
39  class Fish(object):
40      pass
41
42  ## ??
43  class Salmon(Fish):
44      pass
45
46  ## ??
47  class Halibut(Fish):
48      pass
49
50
51  ## rover is-a Dog
52  rover = Dog("Rover")
53
54  ## ??
55  satan = Cat("Satan")
56
57  ## ??
58  mary = Person("Mary")
59
60  ## ??
61  mary.pet = satan
62
63  ## ??
64  frank = Employee("Frank", 120000)
```

```
65
66  ## ??
67  frank.pet = rover
68
69  ## ??
70  flipper = Fish()
71
72  ## ??
73  crouse = Salmon()
74
75  ## ??
76  harry = Halibut()
```

关于class Name(object)

记得我曾经强迫让你使用class Name(object)却没告诉你为什么吧，现在你已经知道了“类”和“对象”的区别，我就可以告诉你原因了。如果我早告诉你，你可能会晕，也学不会这门技术了。

真正的原因是在Python早期，它对于类的定义在很多方面都是严重有问题的。当他们承认这一点的时候已经太迟了，所以逼不得已，他们需要支持这种有问题的类。为了解决已有的问题，他们需要引入一种“新类”，这样的话“旧类”还能继续使用，而你也有一个新的正确的类可以使用了。

这就用到了“类即是对象”的概念。他们决定用小写的“object”这个词作为一个类，让你在创建新类时从它继承下来。有点晕了吧？一个类从另一个类继承，而后者虽然是个类，但名字却叫“object”.....不过在定义类的时候，别忘记要从object继承就好了。

的确如此。一个词的不同就让这个概念变得更难理解，让我不得不现在才讲给你。现在你可以试着去理解“一个是对象的类”这个概念了，如果你感兴趣的话。

不过我还是建议你别去理解了，干脆完全忘记旧格式和新格式类的区别吧，就假设Python要求你在制造类的时候总是要加上(object)，你的脑力要留着思考更重要的问题。

附加练习

- 1.研究一下为什么Python添加了这个奇怪的叫做object的类，它究竟有什么含义呢？
- 2.有没有办法把类当作object使用呢？
- 3.在习题中为animals、fish和people添加一些函数，让它们做一些事情。看看当函数在Animal这样的“基类”（base class）里和在Dog里有什么区别。
- 4.找些别人的代码，理清里边的“是什么”和“有什么”的关系。
- 5.使用列表和字典创建一些新的一对多的“有多个”（has-many）的关系。
- 6.你认为会有一种“有多个”关系吗？阅读一下关于“多重继承”（multiple inheritance）的资料，然后尽量避免这种用法。

常见问题回答

这些`## ??`注释是干嘛用的？

这些注释是供你填空的。你应该在对应的位置填入is-a、has-a的概念。重读这个习题，看看其他的注释，仔细理解一下我的意思。

这句`self.pet = None`有什么用？

确保类的`self.pet`属性被设置为None。

`super(Employee, self).__init__(name)`是做什么用的？

这样你可以可靠地将父类的`__init__`方法运行起来。搜索“python super”，看看它的优缺点。

[1].为了解释方便，译文使用了中文鱼名。原文使用的是“三文鱼”（salmon）和“大比目鱼”（halibut），也是英文常用人名。——译者注

习题43 基本的面向对象分析和设计

我将会讲到用 Python，尤其是通过面向对象编程（OOP）方式实现一些东西的流程。所谓按照流程就是我将给你一系列需要你遵循的步骤，但是并不意味着针对每个不同的问题都要用这个步骤，也并不意味着这个步骤总是能起作用。这些步骤只是解决很多编程问题的一个很好的起点，并不是解决这类问题的唯一方法。这个过程只是你可以遵循的一种方法。

具体过程如下。

- 1.把要解决的问题写下来，或者画出流程图。
- 2.将第一条中的关键概念摘录出来并加以研究。
- 3.创建一个类和对象的层次结构图。
- 4.用代码实现各个类，并写一个测试来运行它们。
- 5.重复上述步骤并细化代码。

这种流程可以看做是一个“自顶向下”（top down）的流程，也就是说从很抽象的概念入手，逐渐细化，直到概念变成具体的可用代码实现的东西。

首先我会把要解决的问题写下来，尽可能想出所有相关的东西。也许我还会画一两张流程图，也许是画些结构关系图，或者给自己写一些描述问题的邮件。这样可以让我把问题的关键概念表达出来，而且还能让我探索自己对这个问题已知的各个方面。

然后我过一遍这些笔记和图示，把里边的关键概念拉出来。有个简单的方法做这件事：把写下的内容里的名词和动词列出来，然后写出它

们之间的关系。这样就会有一份类、对象、函数的名称列表以供下一步使用了。再研究一下这份概念列表中不清楚的部分，这样以后还能在需要的时候进一步细化。

有了关键概念的列表以后，我再为这些概念作为类创建一个结构图（树），把它们之间的关系整理清楚。你可以把名词列表拿出来并问自己：“这里哪些是类似的？意味着它们能用同一个父类，父类又是哪个呢？”到最后你会得到一个类的层次结构，要么是一个简单的树状结构，要么是一张简单的图。然后把动词拿出来，看看它们是不是对应类的函数名称，把它们放到树（图）的对应类的下面。

有了这个层次结构图，我就坐下来写一些基本的骨架代码，里边只包含上面提到的类和函数，没有别的。然后我写一个测试，来保证我写的类是合理的而且能正常工作。有时我会先写测试再写类，有时我会写一点测试再写一点代码，再写一点测试……直到完成整项工作为止。

最后，不断重复这个流程，逐步细化代码，让它在实现更多功能的时候还尽可能保证清晰。如果遇到之前没想到的问题卡在了某处，我就会坐下来按照上面的流程走一遍，等弄清楚了再继续。

现在将用这个流程实现一个游戏引擎和一个游戏。

简单游戏引擎的分析

我想做的游戏名叫《来自Percal 25号行星的哥顿人》，这是一款空间冒险游戏。在我的脑海中只有这个概念，我可以沿着这个思路，完成这款游戏。

把问题写下来或者画出来

我将为这个游戏写一段描述：“外星人入侵了宇宙飞船，我们的英雄需要通过各种房间组成的迷宫，打败遇到的外星人，这样才能通过救生船回到下面的行星上去。这个游戏会跟Zork或者 Adventure 类似，会用文字输出各种搞笑的死法。游戏会用到一个引擎，它带动一张充满房间和场景的地图。当玩家进入一个房间时，这个房间会显示出自己的描述，并且会告诉引擎下一步应该到哪个房间去。”

到目前为止我已经对游戏的内容和运行方式有了一个很好的概念，接下来要描述各个场景。

死亡（**Death**）。玩家死去的场景，应该比较搞笑。

中央走廊（**Central Corridor**）。这是游戏的起点，哥顿人已经在那里把守着，玩家需要讲一个笑话才能继续。

激光武器库（**Laser Weapon Armory**）。在这里英雄会找到一个中子弹，在乘坐救生船离开时要用它把飞船炸掉。这个房间有一个数字键盘，需要猜测密码组合。

飞船主控舱（**The Bridge**）。另一个战斗场景，英雄需要在这里埋炸弹。

救生舱（**Escape Pod**）。英雄逃离的场景，但需要猜对是那艘救生

船。

至此，我应该已经画出了它们的关系图，可能还要为每个房间写更详细的描述。

摘录和研究关键概念

现在有了足够的信息来摘录一些名词，并分析它们的类层次结构。首先整理一个名词列表。

外星人 (Alien)

玩家 (Player)

飞船 (Ship)

迷宫 (Maze)

房间 (Room)

场景 (Scene)

哥顿人 (Gothon)

救生舱 (Escape Pod)

行星 (Planet)

地图 (Map)

引擎 (Engine)

死亡 (Death)

中央走廊 (Central Corridor)

激光武器库 (Laser Weapon Armory)

主控舱 (The Bridge)

我可能还需要把动词摘录出来并看看它们是不是适合做函数名，不过我暂时先跳过这步。

到现在为止你也许已经研究过这些概念以及其中没弄明白的部分了。例如，我可能会通过玩一些类似的游戏来确认它们的工作方式；我

也许会去研究飞船是怎样设计的，以及炸弹是怎样工作的；也许我还会研究一些技术细节，比如怎样把游戏状态存到数据库里去。等完成这些研究后也许需要回到第一步，基于学到的新东西重写游戏描述以及重新摘录相关概念。

为各种概念创建类层次结构图和对象关系图

完成上面的工作后，我会通过问问题的方式把它转成一个类的层次结构图。问题可以是“和其他东西有哪些类似？”或者“哪个只不过是某个东西的另一种叫法？”

很快我就发现，“房间”和“场景”基本上是同一个东西。在这个游戏里，我将使用“场景”，然后我发现“中央走廊”是一个场景，“死亡”也基本上是一个场景。“死亡场景”可以接受，“死亡房间”就有些奇怪了，这也是我选择“场景”这个名词的原因。“迷宫”和“地图”基本上是一个意思，就用“地图”吧，因为这个词平时用的多。这里我不打算实现一个作战系统，所以“外星人”和“玩家”我就先忽略了，留待以后再说。“行星”其实也可以是另一个场景，而不是什么特殊的東西。

理清思路后我开始在文本编辑器里画一个类的层次结构图。

- * Map
- * Engine
- * Scene
 - * Death
 - * Central Corridor
 - * Laser Weapon Armory
 - * The Bridge
 - * Escape Pod

然后我需要查看描述里的动词部分，从而知道每一样东西上面需要

什么样的动作。例如，我需要方法来运行游戏引擎，在地图里转到下一场景，获得初始场景，以及进入一个场景。加上去后大致是下面这样的：

```
* Map
  - next_scene
  - opening_scene
* Engine
  - play
* Scene
  - enter
* Death
* Central Corridor
* Laser Weapon Armory
* The Bridge
* Escape Pod
```

注意，我只在Scene的下面添加了enter这个方法，因为我知道具体的场景会继承并覆盖这个方法。

[编写和运行各个类](#)

准备好了类和函数的树状结构，我需要在编辑器里打开一个源文件，并为它编写代码。通常我只要把这个树状结构复制到源文件中，把它扩写成各个类就可以了。这里是一个初始的简单例子，文件最后还包含一点简单测试。

ex43_classes.py

```
1 class Scene(object):
2
```

```
3         def enter(self):
4             pass
5
6
7     class Engine(object):
8
9         def __init__(self, scene_map):
10             pass
11
12         def play(self):
13             pass
14
15     class Death(Scene):
16
17         def enter(self):
18             pass
19
20     class CentralCorridor(Scene):
21
22         def enter(self):
23             pass
24
25     class LaserWeaponArmory(Scene):
26
27         def enter(self):
28             pass
29
```

```
30 class TheBridge(Scene):
31
32     def enter(self):
33         pass
34
35 class EscapePod(Scene):
36
37     def enter(self):
38         pass
39
40
41 class Map(object):
42
43     def __init__(self, start_scene):
44         pass
45
46     def next_scene(self, scene_name):
47         pass
48
49     def opening_scene(self):
50         pass
51
52
53 a_map = Map('central_corridor')
54 a_game = Engine(a_map)
55 a_game.play()
```

你能看出在这个文件里我只是简单地重复了架构图的内容，然后写

了一点点代码在最后几行，看它是不是能正常工作。后面的几个小节中，你需要把剩下的代码填进去，让它按照上面的游戏描述工作起来。

重复和优化

我的这个小流程的最后一步其实也不算一步，而是一个 `while` 循环。前面所讲的东西并不是一次性操作，需要回去重复整个流程，基于你从后面步骤中学到的东西来优化你写的内容。有时，我走到第三步就会发现需要回到第一步和第二步去弄些东西，那就会停下来回到前面去弄完。有时我会突然来了灵感，然后趁热打铁直接跳到后面把代码写出来，不过接着我会回到前面的步骤来检查并确认我的代码是不是覆盖了所有的可能性。

关于这个流程，要注意的另一个点是，你不需要把自己锁定在一个层面上去完成某个特定任务。假如说不知道怎样写 `Engine.play` 这个方法，可以停下来，就在这个任务上使用这个流程，直到弄明白怎样写为止。

自顶向下与自底向上

我刚描述的流程一般叫做“自顶向下”，因为它是从最抽象的概念（顶层）下手，一直向下做到具体的代码实现。我希望你在继续后面的练习时用这种方法分析问题，不过你应该知道还有一种解决编程问题的方法，就是先从代码下手，一直向上做到抽象概念，这种方法叫做“自底向上”。一般步骤如下。

- 1.取出要解决的问题中的一小块，写些代码让它差不多能工作。
- 2.细化代码让它更为正式，比如加上类和自动测试。
- 3.把关键概念抽取出来然后研究它们。
- 4.把真正需要实现的东西描述出来。
- 5.回去细化代码，有可能需要全部丢弃重头做起。
- 6.在问题的另外一小块里重复上述流程。

我觉得这个流程对于基础牢固的程序员来说更好使，而且也是为解决问题写代码时的自然想法。当你知道一个大问题的小部分，但对于整个总体概念也许还没有足够了解的时候，这种流程是非常好用的。在将问题拆成小块，并且一块一块地解决的过程中，可以慢慢了解问题的方向并且解决它。不过要记住，刚开始写的解决方案可能会走弯路或者很怪异，这就是为什么流程中有一步是回去研究并且基于自己学到的东西把代码细化好。

《来自Percal 25号行星的哥顿人》的代码

停！接下来我将演示上述问题的最终解决方案，不过我不要求你马上下手把它们都写出来。我需要你利用前面写的骨架代码，为它添加功能直到它能实现我们需要的功能。等你实现完以后，再回来看我是怎么实现的。

我就不一下把所有代码贴出来了。我将把这个最终的 `ex43.py` 拆成小块，然后一次解释一块。

`ex43.py`

```
1 from sys import exit
2 from random import randint
```

这就是基本的导入，没什么特别新奇的。

`ex43.py`

```
1 class Scene(object):
2
3     def enter(self):
4         print "This scene is not yet configured.Subclass it and
implement enter()."
5         exit(1)
```

和在骨架代码中看到的一样，有一个叫 `Scene` 的基类，它会包含所有场景的通用信息。这个简单程序里，这些场景并没有多么复杂，所以这基本上只是一个怎样创建基类的演示而已。

`ex43.py`

```
1 class Engine(object):
```

```

2
3     def __init__(self, scene_map):
4         self.scene_map = scene_map
5
6     def play(self):
7         current_scene = self.scene_map.opening_scene()
8
9         while True:
10             print "\n-----"
11             next_scene_name = current_scene.enter()
12             current_scene = self.scene_map.next_scene(next_scene_name)

```

这里我创建好了Engine类，我用了Map.opening_scene和Map.next_scene这些方法，因为这些是我计划好要写的方法，所以我就假设它们已经写好了，这里只是拿来使用。至于Map类，其实我后面才会去写它。

ex43.py

```

1  class Death(Scene):
2
3      quips = [
4          "You died.  You kinda suck at this.",
5          "Your mom would be proud...if she were smarter.",
6          "Such a luser.",
7          "I have a small puppy that's better at this."
8      ]
9
10     def enter(self):

```

```
11         print Death.quips[randint(0, len(self.quips)-1)]
12         exit(1)
```

我写的第一个场景就是这个奇怪的Death场景，这也是最简单的一个场景了。

ex43.py

```
1  class CentralCorridor(Scene):
2
3      def enter(self):
4          print "The Gothons of Planet Percal #25 have invaded your
ship and destroyed"
5          print "your entire crew.  You are the last surviving member
and your last"
6          print "mission is to get the neutron destruct bomb from the
Weapons Armory,"
7          print "put it in the bridge, and blow the ship up after getting
into an "
8          print "escape pod."
9          print "\n"
10         print "You're running down the central corridor to the
Weapons Armory when"
11         print "a Gothon jumps out, red scaly skin, dark grimy teeth,
and evil clown costume"
12         print "flowing around his hate filled body.  He's blocking the
door to the"
13         print "Armory and about to pull a weapon to blast you."
14
15         action = raw_input("> ")
```

```
16
17     if action == "shoot!":
18         print "Quick on the draw you yank out your blaster and fire it
at the Gothon."
19         print "His clown costume is flowing and moving around his
body, which throws"
20         print "off your aim.  Your laser hits his costume but misses
him entirely.  This"
21         print "completely ruins his brand new costume his mother
bought him, which"
22         print "makes him fly into a rage and blast you repeatedly in
the face until"
23         print "you are dead.  Then he eats you."
24         return 'death'
25
26     elif action == "dodge!":
27         print "Like a world class boxer you dodge, weave, slip and
slide right"
28         print "as the Gothon's blaster cranks a laser past your head."
29         print "In the middle of your artful dodge your foot slips and
you"
30         print "bang your head on the metal wall and pass out."
31         print "You wake up shortly after only to die as the Gothon
stomps on"
32         print "your head and eats you."
33         return 'death'
34
```

```

35     elif action == "tell a joke":
36         print "Lucky for you they made you learn Gothon insults in
the academy."
37         print "You tell the one Gothon joke you know:"
38         print "Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr,
fur fvgf nebhaq gur ubhfr."
39         print "The Gothon stops, tries not to laugh, then busts out
laughing and can't move."
40         print "While he's laughing you run up and shoot him square in
the head"
41         print "putting him down, then jump through the Weapon
Armory door."
42         return 'laser_weapon_armory'
43
44     else:
45         print "DOES NOT COMPUTE!"
46         return 'central_corridor'

```

CentralCorridor 是游戏的初始位置，我现在把它做好了。接下来我需要在创建 Map前把其他场景做好，因为在后面的代码中需要引用这些场景。

ex43.py

```

1  class LaserWeaponArmory(Scene):
2
3      def enter(self):
4          print "You do a dive roll into the Weapon Armory,
crouch and scan the room"
5          print "for more Gothon's that might be hiding.  It's dead

```

quiet, too quiet."

```
6         print "You stand up and run to the far side of the room  
and find the"
```

```
7         print "neutron bomb in its container.  There's a keypad  
lock on the box"
```

```
8         print "and you need the code to get the bomb out.  If  
you get the code"
```

```
9         print "wrong 10 times then the lock closes forever and  
you can't"
```

```
10        print "get the bomb.  The code is 3 digits."
```

```
11        code = "%d%d%d" % (randint(1,9), randint(1,9),  
randint(1,9))
```

```
12        guess = raw_input("[keypad]> ")
```

```
13        guesses = 0
```

```
14
```

```
15        while guess != code and guesses < 10:
```

```
16            print "BZZZZEDDD!"
```

```
17            guesses += 1
```

```
18            guess = raw_input("[keypad]> ")
```

```
19
```

```
20        if guess == code:
```

```
21            print "The container clicks open and the seal breaks,  
letting gas out."
```

```
22            print "You grab the neutron bomb and run as fast as  
you can to the"
```

```
23            print "bridge where you must place it in the right  
spot."
```



```
24         return 'the_bridge'
25     else:
26         print "The lock buzzes one last time and then you
hear a sickening"
27         print "melting sound as the mechanism is fused
together."
28         print "You decide to sit there, and finally the
Gothons blow up the"
29         print "ship from their ship and you die."
30         return 'death'
31
32
33
34 class TheBridge(Scene):
35
36     def enter(self):
37         print "You burst onto the Bridge with the netron
destruct bomb"
38         print "under your arm and surprise 5 Goths who are
trying to"
39         print "take control of the ship. Each of them has an
even uglier"
40         print "clown costume than the last. They haven't
pulled their"
41         print "weapons out yet, as they see the active bomb
under your"
42         print "arm and don't want to set it off."
```

```
43
44         action = raw_input("> ")
45
46         if action == "throw the bomb":
47             print "In a panic you throw the bomb at the group
of Gothons"
48             print "and make a leap for the door. Right as
you drop it a"
49             print "Gothon shoots you right in the back killing
you."
50             print "As you die you see another Gothon
frantically try to disarm"
51             print "the bomb.You die knowing they will
probably blow up when"
52             print "it goes off."
53             return 'death'
54
55         elif action == "slowly place the bomb":
56             print "You point your blaster at the bomb under
your arm"
57             print "and the Gothons put their hands up and
start to sweat."
58             print "You inch backward to the door, open it, and
then carefully"
59             print "place the bomb on the floor, pointing your
blaster at it."
60             print "You then jump back through the door,
```

punch the close button"

```
61             print "and blast the lock so the Gothons can't get  
out."
```

```
62             print "Now that the bomb is placed you run to the  
escape pod to"
```

```
63             print "get off this tin can."
```

```
64             return 'escape_pod'
```

```
65         else:
```

```
66             print "DOES NOT COMPUTE!"
```

```
67             return "the_bridge"
```

```
68
```

```
69
```

```
70     class EscapePod(Scene):
```

```
71
```

```
72         def enter(self):
```

```
73             print "You rush through the ship desperately trying to  
make it to"
```

```
74             print "the escape pod before the whole ship  
explodes. It seems like"
```

```
75             print "hardly any Gothons are on the ship, so your run  
is clear of"
```

```
76             print "interference. You get to the chamber with the  
escape pods, and"
```

```
77             print "now need to pick one to take. Some of them  
could be damaged"
```

```
78             print "but you don't have time to look. There's 5  
pods, which one"
```

```
79         print "do you take?"
80
81         good_pod = randint(1,5)
82         guess = raw_input("[pod #]> ")
83
84
85         if int(guess) != good_pod:
86             print "You jump into pod %s and hit the eject
button." % guess
87             print "The pod escapes out into the void of space,
then"
88             print "implodes as the hull ruptures, crushing your
body"
89             print "into jam jelly."
90             return 'death'
91         else:
92             print "You jump into pod %s and hit the eject
button." % guess
93             print "The pod easily slides out into space heading
to"
94             print "the planet below.  As it flies to the planet,
you look"
95             print "back and see your ship implode then
explode like a"
96             print "bright star, taking out the Gothon ship at
the same"
97             print "time.  You won!"
```

98

99

100 return 'finished'

这就是游戏场景的剩余部分了，由于这些场景都是计划好的，代码也来得相当直接。

顺便讲一下，不要直接把这些代码都录进去。记得我说过，试着一点一点地完成。这里只是为了演示最终结果而已。

ex43.py

```
1  class Map(object):
2
3      scenes = [
4          'central_corridor': CentralCorridor(),
5          'laser_weapon_armory': LaserWeaponArmory(),
6          'the_bridge': TheBridge(),
7          'escape_pod': EscapePod(),
8          'death': Death()
9      ]
10
11     def __init__(self, start_scene):
12         self.start_scene = start_scene
13
14     def next_scene(self, scene_name):
15         return Map.scenes.get(scene_name)
16
17     def opening_scene(self):
18         return self.next_scene(self.start_scene)
```

以上就完成了 `Map` 类，你可以看到它把每个场景的名称存在一个

字典中，然后我用`Map.scenes`来调用这些场景。这也是我为什么先写各个场景后写`Map`的原因，因为字典能引用的东西必须是事先存在的。

ex43.py

```
1  a_map = Map('central_corridor')
2  a_game = Engine(a_map)
3  a_game.play()
```

这样整个游戏就完成了。`Map`已经做好，然后把它传到`Engine`里去，再运行`play`，游戏就能正常运行了。

应该看到的结果

首先确认自己弄明白了游戏要实现的东西，并且自己先试着去实现了它。如果实现过程中遇到一些问题，可以偷偷看看我的代码，明白后再回去继续自己的实现。总之要自己先努力尝试过。

我的游戏运行起来是下面这样的。

习题43 会话

```
$ python ex43.py
```

```
-----
```

The Gothons of Planet Percal #25 have invaded your ship and destroyed your entire crew. You are the last surviving member and your last

mission is to get the neutron destruct bomb from the Weapons Armory, put it in the bridge, and blow the ship up after getting into an escape pod.

You're running down the central corridor to the Weapons Armory when a Gothon jumps out, red scaly skin, dark grimy teeth, and evil clown costume flowing around his hate filled body. He's blocking the door to the Armory and about to pull a weapon to blast you.

```
> dodge!
```

Like a world class boxer you dodge, weave, slip and slide right as the Gothon's blaster cranks a laser past your head.

In the middle of your artful dodge your foot slips and you bang your head on the metal wall and pass out.

You wake up shortly after only to die as the Gothon stomps on your

head and eats you.

I have a small puppy that's better at this.

附加练习

- 1.我的代码有个bug，为什么门锁的密码要猜11次而不是10次？
- 2.解释一下房间切换的原理。
- 3.为难度大的房间添加通过的秘籍，我用一行代码两个词就能做出来。
- 4.回到我的描述和分析部分，为英雄和哥顿人创建一个简单的格斗系统。
- 5.这其实是一个小版本的“有限状态机”（finite state machine），找资料阅读了解一下，虽然你可能看不懂，但还是找来看看吧。

常见问题回答

怎样设计自己的游戏故事？

你可以自己编故事，就像跟朋友讲故事一样，也可以从书籍或者电影里找些简单场景，试着实现一下这些自编的场景。

习题44 继承与合成

童话里经常会看到英雄打败恶人的故事，而且故事里总会有一个类似黑暗森林的场景——要么是一个山洞，要么是一片森林，要么是另一个星球，反正是英雄不该去的某个地方。当然，一旦反面角色在剧情中出现，英雄就非得去那片破森林去杀掉坏人。当英雄的总是不得不冒着生命危险进到邪恶森林中去。

你很少会遇到这样的童话故事，说是英雄机智地躲过这些危险处境。你从不会听英雄说：“等等，如果我把公主Buttercup留在家，自己跑出去当英雄闯世界，万一我半路死了，Buttercup就只能嫁给Humperdinck 这个丑八怪王子。Humperdinck 啊，我的老天！我还是呆在这儿，做点出租童工的生意吧。”如果他选择了这条路，就不会遇到火沼泽、死亡、复活、格斗、巨人，或者任何算得上故事的东西了。就是因为这个，这些故事里的森林就像黑洞一样，不管英雄是干嘛的，最终都无法避免地陷入其中。

在面向对象编程中，“继承”（inheritance）就是那片邪恶森林。有经验的程序员知道如何躲开这个恶魔，因为他们知道，在丛林深处的“继承”，其实是邪恶女皇“多重继承”。她喜欢用自己的巨口尖牙吃掉程序员和软件，咀嚼这些堕落者的血肉。不过这片丛林的吸引力是如此强大，几乎每一个程序员都会进去探险，梦想着提着邪恶女皇的头颅走出丛林，从而声称自己是真正的程序员。你就是无法阻止丛林的魔力，于是你深入其中，而等冒险结束，九死一生之后，你唯一学到的就是远远躲开这片森林，而如果你不得不再进去一次，你会带一支军队。

这段故事就是为了教你避免使用“继承”这东西，这样说是不是更有感觉呢？有的程序员现在正在丛林里跟邪恶女皇作战，他会对你说你必须进到森林里去。他们这样说其实是因为他们需要你的帮助，因为他们已经无法承受自己创建的东西了。而对于你来说，你只要记住这一条：大部分使用继承的场合都可以用合成取代，而多重继承则需要不惜一切地避免之。

什么是继承

继承就是用来指明一个类的大部分或全部功能都是从一个父类中获得的。写 `class Foo(Bar)`时，代码就发生了继承效果，这行代码的意思是“创建一个叫Foo的类，并让它继承Bar”。当你这样写时，Python语言会让Bar的实例所具有的功能都工作在Foo的实例上。这样可以让你把通用的功能放到Bar里边，然后再给Foo特别设定一些功能。

当你这么做的时候，父类和子类有三种交互方式：

- 1.子类上的动作完全等同于父类上的动作；
- 2.子类上的动作完全覆盖了父类上的动作；
- 3.子类上的动作部分替换了父类上的动作。

我将通过代码向你一一展示。

隐式继承

首先我将向你展示，当你在父类里定义了一个函数但没有在子类中定义的例子时，会发生隐式继承（implicit inheritance）。

ex44a.py

```
1 class Parent(object):
2
3     def implicit(self):
4         print "PARENT implicit()"
5
6 class Child(Parent):
7     pass
```

```
8
9  dad = Parent()
10 son = Child()
11
12 dad.implicit()
13 son.implicit()
```

class Child:中的 `pass` 是在 Python 中创建空代码块的方法。这样就创建了一个叫Child的类，但没有在里边定义任何细节。在这里它将会从它的父类继承所有的行为。运行起来就是下面这样：

习题44a 会话

```
$ python ex44a.py
PARENT implicit()
PARENT implicit()
```

就算我在第16行调用了`son.implicit()`并且在Child中没有定义过`implicit`这个函数，这个函数依然可以工作，而且和在父类Parent中定义的行为一样。这就说明，如果将函数放到基类中（也就是这里的Parent），那么所有的子类（也就是Child这样的类）将会自动获得这些函数功能。需要很多类的时候，这样可以避免重复写很多代码。

显式覆盖

有时候需要让子类里的函数有一个不同的行为，这种情况下隐式继承是做不到的，而需要覆盖子类中的函数，从而实现它的新功能。只要在子类Child中定义一个相同名称的函数就可以了。下面就是一个例子。

ex44b.py

```
1  class Parent(object):
```

```
2
3     def override(self):
4         print "PARENT override()"
5
6 class Child(Parent):
7
8     def override(self):
9         print "CHILD override()"
10
11 dad = Parent()
12 son = Child()
13
14 dad.override()
15 son.override()
```

这个例子中，我在两个类中都定义了一个叫`override`的函数，我们看看运行时会出现什么情况。

习题44b 会话

```
$ python ex44b.py
PARENT override()
CHILD override()
```

如你所见，运行到第14行时，这里执行的是`Parent.override`，因为`dad`这个变量是定义在`Parent`里的。不过到了第15行，打印出来的却是`Child.override`里的信息，因为`son`是`Child`的一个实例，而子类中新定义的函数在这里取代了父类里的函数。

现在来休息一下并巩固一下这两个概念，然后再接着进行。

[在运行前或运行后替换](#)

使用继承的第三种方法是一个覆盖的特例，这种情况下，你想在父类中定义的内容运行之前或者之后再修改行为。首先像上例一样覆盖函数，不过接着用Python的内置函数super来调用父类Parent里的版本。我们还是来看例子吧。

ex44c.py

```
1 class Parent(object):
2
3     def altered(self):
4         print "PARENT altered()"
5
6 class Child(Parent):
7
8     def altered(self):
9         print "CHILD, BEFORE PARENT altered()"
10        super(Child, self).altered()
11        print "CHILD, AFTER PARENT altered()"
12
13 dad = Parent()
14 son = Child()
15
16 dad.altered()
17 son.altered()
```

重要的是Child中的第9~11行，当调用son.altered()时，我完成了以下内容。

- 1.由于我覆盖了Parent.altered，实际运行的是Child.altered，所以第9行执行结果是预料之中的。

- 2.这里我想在前面和后面加一个动作，所以，第 9 行之后，我要用

`super` 来获取`Parent.altered`这个版本。

3.第10行调用了`super(Child, self).altered()`，这和过去用过的`getattr`很相似，不过它还知道你的继承关系，并且会访问到`Parent`类。这句你可以读作：“用`Child`和`self`这两个参数调用`super`，然后在此返回的基础上调用`altered`。”

4.到这里`Parent.altered`就会被运行，而且打印出了父类里的信息。

5.最后从`Parent.altered`返回到`Child.altered`，函数接着打印出来后面的信息。

运行的结果是下面这样的。

习题44c 会话

```
$ python ex44c.py
PARENT altered()
CHILD, BEFORE PARENT altered()
PARENT altered()
CHILD, AFTER PARENT altered()
```

三种方式组合使用

为了演示上面讲的内容，我来写一个最终版本，在一个文件中演示三种交互模式。

ex44d.py

```
1  class Parent(object):
2
3      def override(self):
4          print "PARENT override()"
5
6      def implicit(self):
7          print "PARENT implicit()"
8
9      def altered(self):
10         print "PARENT altered()"
11
12  class Child(Parent):
13
14      def override(self):
15          print "CHILD override()"
16
17      def altered(self):
18          print "CHILD, BEFORE PARENT altered()"
19          super(Child, self).altered()
```

```

20             print "CHILD, AFTER PARENT altered()"
21
22  dad = Parent()
23  son = Child()
24
25  dad.implicit()
26  son.implicit()
27
28  dad.override()
29  son.override()
30
31  dad.altered()
32  son.altered()

```

回到代码中，在每一行的上方写一个注释，写出它的功能，并且标出它是不是一个覆盖动作，然后运行代码，看看输出的是不是预期的内容。

习题44d 会话

```

$ python ex44d.py
PARENT implicit()
PARENT implicit()
PARENT override()
CHILD override()
PARENT altered()
CHILD, BEFORE PARENT altered()
PARENT altered()
CHILD, AFTER PARENT altered()

```

为什么要用super()

到这里也算是一切正常吧，不过接下来就要来应对一个叫“多重继承”的麻烦东西了。多重继承是指定义的类继承了多个类，就像这样：

```
class SuperFun(Child, BadStuff):  
    pass
```

这相当于说“创建一个叫SuperFun的类，让它同时继承Child和BadStuff”。

这里一旦在SuperFun的实例上调用任何隐式动作，Python就必须回到类的层次结构中去检查Child和BadStuff，而且必须要用固定的次序去检查。为实现这一点Python使用了一个叫“方法解析顺序”（Method Resolution Order, MRO）的东西，还用了一个叫C3的算法。

由于有这个复杂的MRO和这个很好的算法，Python总不该把这些事情留给你去做吧，不然你不就跟着头大了？所以 Python 给你这个super()函数，用来在各种需要修改行为的场合为你处理，就像在上面Child.altered 中一样。有了 super()，再也不用担心把继承关系弄糟，因为Python会找到正确的函数。

super()和__init__搭配使用

最常见的super()的用法是在基类的__init__函数中使用。通常这也是唯一可以进行这种操作的地方，在这里你在子类里做了一些事情，然后完成对父类的初始化。下面是一个在Child中完成上述行为的例子。

```
class Child(Parent):  
    def __init__(self, stuff):  
        self.stuff = stuff  
        super(Child, self).__init__()
```

这和上面的Child.altered差别不大，只不过我在__init__里边先设了个变量，然后才用Parent.__init__初始化了Parent。

合成

继承是一种有用的技术，不过还有一种实现相同功能的方法，就是直接使用别的类和模块，而非依赖于继承。回头来看，我们有三种继承的方式，但有两种会通过新代码取代或者修改父类的功能。这其实可以很容易用调用模块里的函数来实现。我们再来个例子。

ex44e.py

```
1  class Other(object):
2
3      def override(self):
4          print "OTHER override()"
5
6      def implicit(self):
7          print "OTHER implicit()"
8
9      def altered(self):
10             print "OTHER altered()"
11
12 class Child(object):
13
14     def __init__(self):
15         self.other = Other()
16
17     def implicit(self):
```

```

18         self.other.implicit()
19
20     def override(self):
21         print "CHILD override()"
22
23     def altered(self):
24         print "CHILD, BEFORE OTHER altered()"
25         self.other.altered()
26         print "CHILD, AFTER OTHER altered()"
27
28 son = Child()
29
30 son.implicit()
31 son.override()
32 son.altered()

```

这里我没有使用Parent这个名称，因为这里不是父类子类的“A是B”的关系，而是一个“A里有B”的关系，这里Child里有一个Other用来完成它的功能。运行的时候，我们可以看到下面这样的输出。

习题44e 会话

```

$ python ex44e.py
OTHER implicit()
CHILD override()
CHILD, BEFORE OTHER altered()
OTHER altered()
CHILD, AFTER OTHER altered()

```

可以看出，Child 和 Other 里的大部分内容是一样的，唯一不同的是我必须定义一个Child.implicit函数来完成它的功能。然后我可以问自

己，这个Other是写成一个类呢，还是直接做一个叫other.py的模块比较好？

继承和合成的应用场合

“继承与合成”的问题说到底还是为了解决关于代码复用的问题。你不想到处都是重复的代码，这样既难看又没效率。继承可以让你在基类里隐含父类的功能，从而解决这个问题，而合成则是利用模块和别的类中的函数调用达到了相同的目的。

如果两种方案都能解决复用的问题，那什么时候该用哪个方案呢？这个问题的答案其实是非常主观的，不过我可以给你三个大体的指引方案。

- 1.不惜一切代价地避免多重继承，因为它带来的麻烦比能解决的问题都多。如果非要用，那得准备好专研类的层次结构，以及花时间去找各种东西的来龙去脉。

- 2.如果有一些代码会在不同位置和场合应用到，那就用合成来把它们做成模块。

- 3.只有在代码之间有清楚的关联，可以通过一个单独的共性联系起来的时候使用继承，或者受现有代码或者别的不可抗拒因素所限非用不可，那也用吧。

然而，不要成为这些规则的奴隶。面向对象编程中要记住的一点是，程序员创建软件包，共享代码，这些都是一种社交习俗。由于这是一种社交习俗，有时可能因为同事的原因，需要打破这些规则。这时候，就需要去观察别人的工作方式，然后去适应这种场合。

附加练习

本节只有一个附加练习，不过这个附加练习很大。去读一读 <http://www.python.org/dev/peps/pep-0008/>并在代码中应用它。你会发现其中有一些东西和本书中的不一样，不过你现在应该能懂得他们的推荐，并在自己的代码中应用这些规范。本书剩下的部分可能有一些没有完全遵循这些规范，不过这是因为有时候遵循规范反而让代码更难懂。我建议你也照做，因为对代码的理解比对风格规范的记忆更为重要。

常见问题回答

怎样更好地自己解决在前面已经提到的新问题？

提高解决问题能力的唯一方法就是自己去努力解决尽可能多的问题。很多时候人们遇到难题就会跑去找人给出答案。当你手头的事情非要完成不可的时候，这样做是没有问题的，不过如果你有时间自己解决的话，那就花时间去解决吧。停下手上的活，专注于你的问题，试着用所有可能的方法去解决，不管最后解决与否都要试到山穷水尽为止。经过这样的过程，找到的答案会让你更为满意，而你解决问题的能力也会提高。

对象是不是就是类的副本？

有的语言里是这样的，如JavaScript。这样的语言叫做原型（prototype）语言，这种语言里的类和对象除了用法以外没多少不同。不过在Python里类其实像是用来创建对象的模板，就跟制作硬币用到的模具一样。

习题45 你来制作一个游戏

你要开始学会自食其力了。通过阅读这本书你应该已经学到了一点，那就是你需要的所有信息网上都有，你只要去搜索就能找到。唯一困扰你的就是如何使用正确的单词进行搜索。学到现在，你在挑选搜索关键字方面应该已经有些感觉了。现在已经是时候了，你需要尝试写一个大的项目，并让它运行起来。

下面是你的需求。

- 1.制作一个截然不同的游戏。
- 2.使用多个文件，并使用import调用这些文件。确认自己知道import的用法。
- 3.每个房间使用一个类，类的命名要能体现出它的用处，如GoldRoom、KoiPondRoom。
- 4.你的运行器代码应该了解这些房间，所以创建一个类来调用并记录这些房间。有很多种方法可以达到这个目的，可以考虑让每个房间返回下一个房间，或者设置一个变量，让它指定下一个房间是什么。

其他事情就全靠你了。花一个星期完成这个任务，做一个你能做出来的最好的游戏。用学过的任何东西（类、函数、字典、列表.....）来改进你的程序。这个习题的目的是教你如何构建能调用其他Python文件中的类的类。

我不会详细告诉你怎样做，你需要自己完成。试着动手吧，编程就是解决问题的过程，这就意味着你要尝试各种可能性，进行实验，经历失败，然后丢掉做出来的东西重头开始。当你被某个问题卡住的时候，

可以向别人寻求帮助，把自己的代码贴出来给他们看。如果有人对你很刻薄，别理他们，你只要集中精力在帮你的人身上就可以了。持续修改和清理你的代码，直到它足够好，然后再研究一下看它还能不能被改进。

祝你好运，下个星期你做出游戏后我们再见。

评价你的游戏

这个习题的目的是检查评估你的游戏。也许你只完成了一半，卡在那里没有进行下去，也许你勉强做出来了。不管怎样，我们将串一下你应该弄懂的一些东西，并确认你的游戏里有用到它们。我们将学习用正确的格式构建类的方法、使用类的一些通用习惯，另外还有很多“书本知识”。

为什么我会让你先尝试然后才告诉你正确的做法呢？因为从现在开始你要学会“自食其力”，以前是我牵着你前行，以后就得靠你自己了。后面的习题我只会告诉你你的任务是什么，你需要自己去完成，你完成后我再告诉你如何可以改进你所做的。

一开始你会觉得很困难并且很不习惯，但只要坚持下去，你就会培养出自己解决问题的能力。你会找出创新的方法解决问题，这比从课本中复制解决方案强多了。

函数的风格

以前我教过的怎样写好函数的方法一样是适用的，不过这里还要添加几条。

由于各种各样的原因，程序员将类里边的函数称作“方法”（`method`）。很大程度上这只是个营销策略（用来推销 OOP），不过如果你把它们称作“函数”，是会有人跳出来纠正你的。如果你觉得他们太烦，可以让他们从数学方面演示一下“函数”和“方法”究竟有什么不同，这样他们就会很快闭嘴。

在使用类的过程中，很大一部分时间是告诉你的类如何“做事情”。给这些函数命名的时候，与其命名成一个名词，不如命名为一个动词，作为给类的一个命令。就和list的 `pop`（弹出）函数一样，它相当于说：“嘿，列表，把这东西给我弹出去。”它的名字不是 `remove_from_end_of_list`，因为即使它的功能的确是这样，这一串字符也不是一个命令。

让函数保持简单小巧。由于某些原因，有些人开始学习类后就会忘了这一条。

类的风格

类应该使用“驼峰式大小写”（camel case），如应该使用 `SuperGoldFactory` 而不是 `super_gold_factory`。

`__init__` 不应该做太多的事情，这会让类变得难以使用。

其他函数应该使用“下划线隔词”，所以可以写 `my_awesome_hair`，而不是 `myawesomehair` 或者 `MyAwesomeHair`。

用一致的方式组织函数的参数。如果类需要处理 `users`、`dogs` 和 `cats`，就保持这个次序（特殊情况除外）。如果一个函数的参数是 `(dog, cat, user)`，另一个的是 `(user, cat, dog)`，这会让函数使用起来很困难。

不要对全局变量或者来自模块的变量进行重定义或者赋值，让这些东西自顾自就行了。

不要一根筋式地维持风格一致性。一致性是好事情，不过愚蠢地跟着别人遵从一些白痴口号是错误的行为——这本身就是一种坏的风格。好好为自己着想吧。

永远都使用 `class Name(object)` 的方式定义类，否则会遇到大麻烦。

代码风格

为了方便他人阅读，为自己的代码字符之间留下一些空白。有些程序员写的代码还算通顺，但字符之间没有任何空间。这种风格在任何编程语言中都是坏习惯，人的眼睛和大脑会通过空白和垂直对齐的位置来扫描和区隔视觉元素，如果你的代码里没有任何空白，这相当于为你的代码上了“迷彩装”。

如果一段代码你无法朗读出来，那么这段代码的可读性可能就有问题。如果你找不到让某个东西易用的方法，试着也朗读出来。这样不仅会逼迫你慢速而且真正仔细阅读，还会帮你找到难读的段落，从而知道那些代码的易读性需要作出改进。

学着模仿别人的风格写Python程序，直到哪天你找到自己的风格为止。

一旦你有了自己的风格，也别把它太当回事儿。程序员工作的一部分就是和别人的代码打交道，有的人审美就是很差。相信我，你的审美某一方面一定也很差，只是你从未意识到而已。

如果你发现有人写的代码风格你很喜欢，那就模仿他们的风格。

好的注释

有程序员会告诉你，说你的代码需要有足够的可读性，这样就无需写注释了。他们会以略带官腔的声音说：“所以你永远都不应该写代码注释。”这些人要么是一些顾问型的人物，如果别人无法使用他们的代码，就会付更多钱给他们，让他们解决问题，要么他们能力不够，从来没有跟别人合作过。别理会这些人，好好写你的注释。

写注释的时候，描述清楚为什么要这样做。代码只会告诉你“这样实现”，而不会告诉你“为什么要这样实现”，而后者比前者更重要。

为函数写文档注释的时候，记得为别的代码使用者也写些东西。不需要狂写一大堆，但一两句话写写这个函数的用法还是很有用的。

最后要说的是，虽然注释是好东西，但太多的注释就不见得是了。而且注释也是需要维护的，要尽量让注释短小精悍、一语中的，如果你对代码做了更改，记得检查并更新相关的注释，确认它们还是正确的。

为你的游戏评分

现在假装你就是我，板起脸来，把你的代码打印出来，然后拿一支红笔，把代码中所有的错误都标出来。要充分利用你在这个习题以及前面的习题中学到的知识。等你批改完了，请把所有的错误改对。这个过程需要你多重复几次，争取找到更多的可以改进的地方。使用前面教过的方法，把代码分解成最细小的单元一一进行分析。

这个习题的目的是训练你对于细节的关注程度。等你检查完自己的代码，再找一段别人的代码，用这种方法检查一遍。把代码打印出来，检查出所有代码和风格方面的错误，然后试着在不改坏别人代码的前提下把它们修改正确。

这周要你的事情就是批改和纠错，包含你自己的代码和别人的代码，再没有别的了。这个习题难度还是挺大的，不过一旦完成了这个任务，你学过的东西就会牢牢记在脑海中。

习题46 项目骨架

这里你将学会如何建立一个项目“骨架”目录。这个骨架目录具备让项目跑起来的所有基本内容。它里边会包含你的项目文件布局、自动测试代码、模块及安装脚本。当你建立一个新项目的时候，只要把这个目录复制过去，改改目录的名字，再编辑里边的文件就行了。

Python软件包的安装

你需要预先安装一些软件包，不过问题就来了。我的本意是让这本书越清晰越干净越好，不过安装软件的方法实在是太多了，如果要一步一步写下来，那10页都写不完。

所以我不会提供详细的安装步骤，只会告诉你需要安装哪些东西，然后你自己搞定。这对你也有好处，因为你将打开一个全新的世界，里边充满了其他人发布的Python软件。

接下来需要安装下面这些软件包。

1.pip: <http://pypi.python.org/pypi/pip>。

2.distribute: <http://pypi.python.org/pypi/distribute>。

3.nose: <http://pypi.python.org/pypi/nose>。

4.virtualenv: <http://pypi.python.org/pypi/virtualenv>。

不要只是手动下载并且安装这些软件包，还应该看一下别人的建议，尤其看看针对你的操作系统别人是怎样建议你安装和使用的。同样的软件包在不一样的操作系统上安装方式是不一样的，不一样版本的Linux和OSX会有不同，而Windows更是不同。

我要预先警告你，这个过程会相当无趣。在业内我们将这种事情叫做“yak shaving”（剃牦牛）。它指的是，在你做一件有意义的事情之前的一些准备工作，而这些准备工作又是及其无聊冗繁的。你要做一个很酷的Python项目，但是创建骨架目录需要你安装一些软件包，而安装软件包之前还要安装软件包安装工具（package installer），而要安装这个工具还得先学会如何在你的操作系统下安装软件，真是烦不胜烦呀。

无论如何，还是克服困难把。就把它当做进入编程俱乐部的一个考

验。每个程序员都会经历这条道路，在每一段“酷”的背后总会有一段“烦”的。

注意 有时候Python安装程序并不添加C:\Python27\Script到系统路径（PATH）中。如果你遇到这种情况，你可以像在习题0中针对C:\Python27所做的那样，使用以下语句将其添加到系统路径中：

```
[Environment]::SetEnvironmentVariable("Path",  
"$env:Path;C:\Python27\Script", "User")
```

创建骨架项目目录

首先使用下述命令创建骨架目录的结构：

```
$ mkdir projects
$ cd projects/
$ mkdir skeleton
$ cd skeleton
$ mkdir bin
$ mkdir NAME
$ mkdir tests
$ mkdir docs
```

我用了一个叫 `projects` 的目录来存储自己的各个项目，然后在里边建立了一个叫做 `skeleton` 的目录，这就是我们新项目的基础目录。其中叫 `NAME` 的目录是你的项目的主模块，使用骨架时，你可以给它任意取名。

接下来要设置一些初始文件。下面是如何在Linux/OSX环境下进行配置：

```
$ touch NAME/__init__.py
$ touch tests/__init__.py
```

在Windows PowerShell中的设置方式如下：

```
$ new-item -type file NAME/__init__.py
$ new-item -type file tests/__init__.py
```

以上命令创建了空模块目录，以供后面为其添加代码。然后我们需要建立一个 `setup.py` 文件，这个文件在安装项目的时候会用到。

setup.py

```
1  try:
2      from setuptools import setup
3  except ImportError:
4      from distutils.core import setup
5
6  config = [
7      'description': 'My Project',
8      'author': 'My Name',
9      'url': 'URL to get it at.',
10     'download_url': 'Where to download it.',
11     'author_email': 'My email.',
12     'version': '0.1',
13     'install_requires': ['nose'],
14     'packages': ['NAME'],
15     'scripts': [],
16     'name': 'projectname'
17 ]
18
19 setup(**config)
```

编辑这个文件，把自己的联系方式写进去，然后放到那里就行了。

最后需要一个简单的测试专用的骨架文件叫tests/NAME_tests.py:

NAME_tests.py

```
1  from nose.tools import *
2  import NAME
3
4  def setup():
```



```

5         print "SETUP!"
6
7     def teardown():
8         print "TEAR DOWN!"
9
10    def test_basic():
11        print "I RAN!"

```

[最终目录结构](#)

完成了一切准备工作时，你的目录看上去应该和我这里的一样：

```

$ ls -R
NAME                                bin                                docs
setup.py                            tests
./NAME:
__init__.py
./bin:
./docs:
./tests:
NAME_tests.py                       __init__.py

```

这是在Unix下看到的东西，不过在Windows下结构也是一样的。如果以树状结构显示就是下面这个样子：

```

setup.py
NAME/
    __init__.py
bin/
docs/

```

```
tests/
```

```
    NAME_tests.py
```

```
    __init__.py
```

从现在开始，你应该在这层目录运行相关的命令。如果你运行 `ls -R` 看到的不是这个目录架构，那你所处的目录就是错的。例如，人们经常到 `tests/` 目录下运行那里的文件，但这样是行不通的。要运行你的测试，你需要到 `tests/` 的上一级目录，也就是这里显示的目录来运行。所以，如果你运行下面的命令就大错特错了！

```
$ cd tests/ # WRONG! WRONG! WRONG!
```

```
$ nosetests
```

```
-----  
Ran 0 tests in 0.000s
```

```
OK
```

你必须在 `tests` 目录的上一层运行才可以，所以假设你犯了这个错误，应该用下面的方法来正确执行：

```
$ cd ..      # get out of tests/
```

```
$ ls         # CORRECT! you are now in the right spot
```

```
NAME                bin                docs
```

```
setup.py            tests
```

```
$ nosetests
```

```
.
```

```
-----  
Ran 1 test in 0.004s
```

```
OK
```

这一条一定要记住，因为人们经常犯这样的错误。

测试你的配置

安装了所有软件包以后，就可以做下面的事情了：

```
$ nosetests
```

```
.
```

```
-----
```

```
Ran 1 test in 0.007s
```

```
OK
```

下一个习题中我会告诉你nosetests的功能，不过如果你没有看到上面的内容，那就说明哪里出错了。确认一下你的 `NAME` 和 `tests` 目录下存在 `__init__.py`，并且你没有把 `tests/NAME_tests.py` 命名错。

使用这个骨架

“剃牦牛”的事情已经做的差不多了，以后每次要新建一个项目时，只要做下面的事情就可以了。

- 1.复制这份骨架目录，把名字改成新项目的名字。
- 2.再将NAME模块更名为你需要的名字，它可以是你的项目的名字，当然别的名字也行。
- 3.编辑setup.py，让它包含新项目的相关信息。
- 4.重命名tests/NAME_tests.py，让它的名字匹配到你的模块的名字。
- 5.使用nosetests检查有无错误。
- 6.开始写代码。

小测验

这个习题没有附加练习，不过需要你做一个小测验。

- 1.找文档阅读，学会使用前面安装了的软件包。
- 2.阅读关于 `setup.py` 的文档，看它里边可以做多少配置。Python 的安装器并不是一个好软件，所以使用起来也非常奇怪。
- 3.创建一个项目，在模块目录里写一些代码，并让这个模块可以运行。
- 4.在bin目录下放一个可以运行的脚本。找材料学习一下怎样创建可以在系统下运行的Python脚本。
- 5.在`setup.py`中加入bin里的脚本，这样你安装时就可以连它安装进去。
- 6.使用`setup.py`安装你的模块，并确定安装的模块可以正常使用，最后使用`pip`将其卸载。

常见问题回答

这些指令在**Windows**下能用吗？

应该可以，不过在某些版本的Windows里可能会遇到一点儿困难。自己去研究并尝试，直到搞定为止，或者找有经验的朋友帮忙也可以。

Windows下好像不能运行**nosetests**？

有时Python安装包不会把C:\Python27\Script加到系统PATH中。如果遇到这种情况，就照着Ex0里的说明把上述路径也加到PATH中。

setup.py的配置字典中该放些什么信息进去？

读读distutils的文档就知道了：

<http://docs.python.org/distutils/setupscript.html>。

没法加载**NAME**模块，遇到了**ImportError**。

确定创建了NAME/__init__.py文件。如果用的是Windows，那就再检查一下是不是被命名成了NAME/__init__.py.txt，有的编辑器会默认弄成这个样子。

为什么非要弄个**bin/**文件夹？

这只是一个标准的位置，用来存放从命令行运行的脚本，但这不是存放模块的地方。

有没有实际项目的代码可以给我看看？

很多Python项目都用了类似的结构，你可以看看我做的这个简单项目：<https://gitorious.org/python-modargs>。

我的**nosetests**只显示运行了一个测试。这样有没有问题？

没问题。我的输出也是这样子的。

习题47 自动化测试

为了确认游戏的功能正常，你需要一遍一遍地在你的游戏中输入命令。这个过程是很枯燥无味的。如果能写一小段代码用来测试代码岂不是更好？然后只要你对程序做了任何修改，或者添加了什么新东西，只要“跑一下测试”，而这些测试能确认程序依然能正确运行。这些自动测试不会抓到所有的bug，但可以让你无需重复输入命令运行你的代码，从而为你节约很多时间。

这个习题后面的习题不会再有“应该看到的结果”这一节了，取而代之的是一个“应该测试的东西”一节。从现在开始，你需要为自己写的所有代码写自动测试，这会让你成为一名更好的程序员。

我不会解释为什么你需要写自动测试。我要告诉你的是，想要成为一名程序员，而程序的作用是让无聊冗繁的工作自动化，测试软件毫无疑问是无聊冗繁的，所以你还是写点代码让它为你测试的好。

因为写单元测试的原因是让代码更加强健，所以这应该是你需要的所有的解释了。你读了这本书，写了很多代码来做一些事情。现在将更进一步，写出懂得你写的其他代码的代码。这个写代码测试你写的其他代码的过程将强迫你清楚地理解你之前写的代码。这会让你更清晰地了解你写的代码实现的功能及其原理，而且让你对细节的注意程度更上一个台阶。

编写测试用例

下面将以一段非常简单的代码为例，写一个简单的测试，这个测试将建立在上一个习题中我们创建的项目骨架上。

首先，从你的项目骨架创建一个叫做ex47的项目。下面是要采取的步骤。我将用英文给出这些指令而不是展示如何录入它们，你必须理解这一点。

- 1.复制骨架到ex47中。
- 2.将带有NAME的东西都重命名为ex47。
- 3.文件中的NAME全部改为ex47。
- 4.删除所有*.py文件，确保已经清理干净。

注意 请记住，你要运行nosetests来运行测试。你可以通过python ex46-tests.py来运行它们，但这不容易工作，你不得不为每个测试文件做一次。

如果你遇到困难，回头看一下习题46。如果完成这些还不是很容易，需要多练习几次。

接下来创建一个简单的 ex47/game.py 文件，里边放一些要测试的代码。现在放一个傻乎乎的小类进去作为测试对象。

game.py

```
1 class Room(object):
2
3     def __init__(self, name, description):
4         self.name = name
5         self.description = description
```



```

6         self.paths = []
7
8     def go(self, direction):
9         return self.paths.get(direction, None)
10
11     def add_paths(self, paths):
12         self.paths.update(paths)

```

准备好了这个文件，接下来把单元测试骨架改成下面这个样子。

ex47_tests.py

```

1  from nose.tools import *
2  from ex47.game import Room
3
4
5  def test_room():
6      gold = Room("GoldRoom",
7                  """This room has gold in it you can
grab. There's a
8                      door to the north.""")
9      assert_equal(gold.name, "GoldRoom")
10     assert_equal(gold.paths, [])
11
12  def test_room_paths():
13      center = Room("Center", "Test room in the center.")
14      north = Room("North", "Test room in the north.")
15      south = Room("South", "Test room in the south.")
16
17      center.add_paths(['north': north, 'south': south])

```

```

18         assert_equal(center.go('north'), north)
19         assert_equal(center.go('south'), south)
20
21     def test_map():
22         start = Room("Start", "You can go west and down a hole.")
23         west = Room("Trees", "There are trees here, you can go
east.")
24         down = Room("Dungeon", "It's dark down here, you can go
up.")
25
26         start.add_paths(['west': west, 'down': down])
27         west.add_paths(['east': start])
28         down.add_paths(['up': start])
29
30         assert_equal(start.go('west'), west)
31         assert_equal(start.go('west').go('east'), start)
32         assert_equal(start.go('down').go('up'), start)

```

这个文件导入了你在ex47.game模块中创建的Room类，接下来要做的就是测试它。于是我们看到一系列以 `test_` 开头的测试函数，它们就是所谓的“测试用例”（test case），每一个测试用例里面都有一小段代码，它们会创建一个或者一些房间，然后去确认房间的功能和你期望的是否一样。它测试了基本的房间功能，然后测试了路径，最后测试了整个地图。

这里最重要的函数是`assert_equal`，它保证了你设置的变量以及你在Room里设置的路径和你的期望相符。如果得到错误的结果，`nosetests`将会打印出一个错误信息，这样就可以找到出错的地方并修正过来了。

测试指南

在写测试代码时，你可以照着下面这些不是很严格的指南来做。

1.测试脚本要放到tests/目录下，并且命名为BLAH_tests.py，否则nosetests就不会执行你的测试脚本了。这样做还有一个好处就是防止测试代码和别的代码互相混掉。

2.为你的每一个模块写一个测试。

3.测试用例（函数）保持简短，但如果看上去不怎么整洁也没关系，测试用例一般都有点乱。

4.就算测试用例有些乱，也要试着让他们保持整洁，把里边重复的代码删掉。创建一些辅助函数来避免重复的代码。当你下次在改完代码需要改测试的时候，你会感谢我这一条建议的。重复的代码会让修改测试变得很难操作。

5.最后一条是别太把测试当做一回事。有时候，更好的方法是把代码和测试全部删掉，然后重新设计代码。

应该看到的结果

习题47 会话

```
$ nosetests
```

```
...
```

```
-----
```

```
Ran 3 tests in 0.008s
```

```
OK
```

如果一切工作正常的话，你看到的结果应该就是这样。试着把代码改错几个地方，然后看错误信息会是什么，再把代码改正确。

附加练习

1. 仔细阅读与nosetest相关的文档，再去了解一下其他的替代方案。
2. 了解一下Python的“doctest”，看看你是不是更喜欢这种测试方式。
3. 改进你游戏里的 Room，然后用它重建你的游戏，这次重写，你需要一边写代码，一边把单元测试写出来。

常见问题回答

运行**nosetests**时出现语法错误。

看看错误信息的具体内容，把对应行的语法错误改正过来。

nosetests 这类工具会运行你写的程序代码和测试代码，所以和Python一样，它也会找出你的语法错误。

无法导入**ex47.game**？

确认你创建了**ex47/__init__.py**文件，回到前面的内容看看如何创建。

运行**nosetests**时看到**UserWarning**。

你也许装了两个版本的 Python，或者你没有使用 **distribute**，照着习题 46 装一下**distribute**或者**pip**就可以了。

习题48 更复杂的用户输入

你的游戏可能已经能工作的很好了，但处理用户输入的方式肯定让你不胜其烦。每一个房间都需要一套自己的语句，而且只有用户输入完全正确后才能执行。你需要一个设备，它允许用户以各种方式输入短语。例如，下面的几种表述都应该被支持：

Open door

Open the door

Go THROUGH the door

下面的两种表述也应该被支持：

Punch bear

Punch The Bear in the FACE

也就是说，如果用户的输入和常用英语很接近也应该是可以的，而你的游戏要识别出它们的意思。为了达到这个目的，需要写一个模块专门做这件事情。这个模块里边会有若干个类，它们互相配合，接收用户输入，并且将用户输入转换成你的游戏可以识别的命令。

英语的简单格式是下面这个样子的：

单词由空格隔开；

句子由单词组成；

语法控制句子的含义。

所以最好的开始方式是先搞定如何得到用户输入的单词，并且判断出它们是什么。

我们的游戏语汇

我在游戏里创建了下面这个单词的语汇表。

表示方向的单词：north、south、east、west、down、up、left、right、back。

动词：go、stop、kill、eat。

修饰词：the、in、of、from、at、it。

名词：door、bear、princess、cabinet。

数词：由0~9构成的数字。

说到名词，我们会遇到一个小问题，那就是不一样的房间会用到不一样的一组名词，不过让我们先挑一小组出来写程序，以后再做改进吧。

断句

我们已经有了单词的语汇表，为了分析句子的意思，接下来需要找到一种断句的方法。我们对于句子的定义是“空格隔开的单词”，所以只要这样就可以：

```
stuff = raw_input('> ')
```

```
words = stuff.split()
```

目前做到这样就可以了，不过这招儿在相当一段时间内都不会有问题。

语汇元组

一旦我们知道了如何将句子断成单词，剩下的就是逐一检查这些单

词，看它们是什么“类型”的。为了达到这个目的，我们将用到一个非常好用的Python数据结构，叫做“元组”（tuple）。元组其实就是一个不能修改的列表。创建它的方法和创建列表差不多，成员之间需要用逗号隔开，不过方括号要换成圆括号（`()`）：

```
first_word = ('direction', 'north')
second_word = ('verb', 'go')
sentence = [first_word, second_word]
```

这样就创建了一个(TYPE, WORD)组，让你识别出单词，并且对它执行指令。

这只是一个例子，不过最后做出来的样子也差不多。你接收用户输入，用 `split` 将其分隔成单词，然后分析这些单词，识别它们的类型，最后重新组成一个句子。

扫描输入

现在你要写的是扫描器。这个扫描器会将用户输入的字符串当做参数，然后返回由多个(TOKEN, WORD)组成的一个列表，这个列表实现类似句子的功能。如果一个单词不在预定的单词语汇表中，那它返回时WORD应该还在，但TOKEN应该设置成一个专门的错误标记。这个错误标记将告诉用户哪里出错了。

有趣的地方来了。我不会告诉你这些该怎样做，但我会写一个“单元测试”（unit test），而你要编写扫描器写出来，以便保证单元测试能够正常通过。

异常和数字

有一件小事情我会先帮帮你，那就是数字转换。为了做到这一点，我们会作一点儿弊，使用“异常”（exception）来做。“异常”指的是运行

某个函数时得到的错误。你的函数在遇到错误时，就会“引发”（raise）一个“异常”，然后你就要去“处理”（handle）这个异常。假如你在Python里写了这些东西：

```
~/projects/simplegame $ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> int("hell")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hell'
>>>
```

这个ValueError就是int()函数抛出的一个异常，因为你给int()的参数不是一个数字。int()函数其实也可以返回一个值来告诉你它遇到了错误，不过由于它只能返回整数值，所以很难做到这一点。它不能返回-1，因为这也是一个数字。int()没有纠结在它“究竟应该返回什么”上面，而是提出了一个叫做ValueError的异常，然后你只要处理这个异常就可以了。

处理异常的方法是使用try和except这两个关键字：

```
def convert_number(s):
    try:
        return int(s)
    except ValueError:
        return None
```

把要试着运行的代码放到try块里，再将出错后要运行的代码放到except块里。在这里，要试着调用int()去处理某个可能是数字的东西，如果中间出了错，就抓到这个错误，然后返回None。

在你写的扫描器里面，你应该使用这个函数来测试某个东西是不是数字。做完这个检查，你就可以声明这个单词是一个错误单词了。

应该测试的东西

下面是你应该使用的测试文件tests/lexicon_tests.py。

ex48.py

```
1  from nose.tools import *
2  from ex48 import lexicon
3
4
5  def test_directions():
6      assert_equal(lexicon.scan("north"), [('direction', 'north')])
7      result = lexicon.scan("north south east")
8      assert_equal(result, [('direction', 'north'),
9                             ('direction', 'south'),
10                            ('direction', 'east')])
11
12  def test_verbs():
13      assert_equal(lexicon.scan("go"), [('verb', 'go')])
14      result = lexicon.scan("go kill eat")
15      assert_equal(result, [('verb', 'go'),
16                             ('verb', 'kill'),
17                             ('verb', 'eat')])
18
19
20  def test_stops():
```

```

21     assert_equal(lexicon.scan("the"), [('stop', 'the')])
22     result = lexicon.scan("the in of")
23     assert_equal(result, [('stop', 'the'),
24                           ('stop', 'in'),
25                           ('stop', 'of')])
26
27
28     def test_nouns():
29         assert_equal(lexicon.scan("bear"), [('noun', 'bear')])
30         result = lexicon.scan("bear princess")
31         assert_equal(result, [('noun', 'bear'),
32                               ('noun', 'princess')])
33
34     def test_numbers():
35         assert_equal(lexicon.scan("1234"), [('number', 1234)])
36         result = lexicon.scan("3 91234")
37         assert_equal(result, [('number', 3),
38                               ('number', 91234)])
39
40
41     def test_errors():
42         assert_equal(lexicon.scan("ASDFADFASDF"), [('error',
43 'ASDFADFASDF')])
44         result = lexicon.scan("bear IAS princess")
45         assert_equal(result, [('noun', 'bear'),
46                               ('error', 'IAS'),
47                               ('noun', 'princess')])

```

记住要使用你的项目骨架来创建新项目，将这个测试用例写下来（不许复制粘贴！），然后编写你的扫描器，直至所有的测试都能通过。注意细节并确认一切工作良好。

设计提示

集中一次实现一个测试项目，尽量保持项目简单，只要把你的 `lexicon.py` 模块中的语汇表的所有单词放那里就可以了。不要修改输入的单词列表，但是要创建自己的新列表，里边包含你的语汇元组。另外，记得使用 `in` 关键字来检查这些语汇列表，以确认某个单词是否在你的语汇表中。在你的解决方案中使用目录。

附加练习

- 1.改进单元测试，让它覆盖到更多的语汇。
- 2.向语汇列表添加更多的单词，并更新单元测试代码。
- 3.确认你的扫描器能够识别任意大小写的单词。更新单元测试以确认它实际工作。
- 4.找出另一种转换数字的方法。
- 5.我的解决方案用了37行代码，你的是更长还是更短呢？

常见问题回答

为什么我老看到**ImportError**?

通常有四样错误会导致 **ImportError**: (1)在模块路径下没有创建 `__init__.py`; (2)在错误的路径下执行了 `import`; (3)拼写错误, 导致导入了错误的模块; (4)没有设置到 `PYTHONPATH`, 所以无法从当前路径加载模块。

try-except和**if-else**有何不同?

try-expect仅用于处理异常, 绝不要把它作为**if-else**使用。

有没有办法让游戏在等待用户输入的时候不间断地运行?

我猜想你是想把游戏做得更高级, 用户反应过慢就被怪物杀死之类的。这个是可以做到的, 不过需要用到更高级的模块和编程技巧, 这些内容本书不会涉及。

习题49 创建句子

从这个小游戏的语汇扫描器中，我们应该可以得到类似下面的列表：

```
>>> from ex48 import lexicon
>>> print lexicon.scan("go north")
[('verb', 'go'), ('direction', 'north')]
>>> print lexicon.scan("kill the princess")
[('verb', 'kill'), ('stop', 'the'), ('noun', 'princess')]
>>> print lexicon.scan("eat the bear")
[('verb', 'eat'), ('stop', 'the'), ('noun', 'bear')]
>>> print lexicon.scan("open the door and smack the bear in the nose")
[('error', 'open'), ('stop', 'the'), ('noun', 'door'), ('error', 'and'),
('error', 'smack'), ('stop', 'the'), ('noun', 'bear'), ('stop', 'in'),
('stop', 'the'), ('error', 'nose')]
>>>
```

现在让我们把它转化成游戏可以使用的东西，也就是一个语句类。
在学校学过，一个句子是由这样的结构组成的：

主语 谓语（动词） 宾语

显然，实际的句子可能会比这复杂，而你可能已经在英语的语法课上被折腾得够呛了。我们的目的是将上面的元组列表转换为一个语句对象，而这个对象又包含主、谓、宾各个成员。

match和peek

要达到这个效果，需要以下四样工具。

- 1.循环访问元组列表的方法，这挺简单的。
- 2.“匹配”（`match`）主谓宾设置中不同种类元组的方法。
- 3.“窥视”（`peek`）潜在元组的方法，以便做决定时用到。
- 4.“跳过”（`skip`）我们不关心的内容的方法，如形容词、冠词等修饰词。

把这些函数放到一个叫`ex48/parser.py`的文件中以方便对其进行测试。我们使用`peek`函数来查看元组列表中的下一个成员，然后使用`match`函数取出一个元素对其进行操作。让我们先看看这个`peek`函数：

```
def peek(word_list):
    if word_list:
        word = word_list[0]
        return word[0]
    else:
        return None
```

很简单。再看看`match`函数：

```
def match(word_list, expecting):
    if word_list:
        word = word_list.pop(0)
        if word[0] == expecting:
            return word
    else:
```

```
        return None
    else:
```

```
        return None
```

还很简单。最后看看skip函数：

```
def skip(word_list, word_type):
    while peek(word_list) == word_type:
        match(word_list, word_type)
```

以你现在的水平应该可以看出它们的功能来了。确认自己真的弄懂了。

句子的文法

有了工具，现在可以从元组列表来构建句子对象了。我们的处理流程具体如下。

- 1.使用`peek`识别下一个单词。
- 2.如果这个单词和我们的语法匹配，就调用一个函数来处理文法的这部分。假设函数的名字叫`parse_subject`好了。
- 3.如果语法不匹配，就产生一个错误，接下来你会学到这方面的内容。
- 4.全部分析完以后，应该能得到一个语句对象，然后可以将其应用在我们的游戏中。

演示这个过程最简单的方法是把代码展示给你，让你阅读，不过这个习题有个不一样的要求，前面是我给出测试代码，你照着写出程序来，而这次是我给你的程序，你为它写出测试代码来。

下面就是我写的用来解析简单句子的代码，它使用了`ex48.lexicon`这个模块。

ex49.py

```
1 class ParserError(Exception):
2     pass
3
4
5 class Sentence(object):
6
7     def __init__(self, subject, verb, object):
```

```
8          # remember we take ('noun','princess') tuples and
convert them
9          self.subject = subject[1]
10         self.verb = verb[1]
11         self.object = object[1]
12
13
14     def peek(word_list):
15         if word_list:
16             word = word_list[0]
17             return word[0]
18         else:
19             return None
20
21
22     def match(word_list, expecting):
23         if word_list:
24             word = word_list.pop(0)
25
26             if word[0] == expecting:
27                 return word
28             else:
29                 return None
30         else:
31             return None
32
33
```

```
34 def skip(word_list, word_type):
35     while peek(word_list) == word_type:
36         match(word_list, word_type)
37
38
39 def parse_verb(word_list):
40     skip(word_list, 'stop')
41
42     if peek(word_list) == 'verb':
43         return match(word_list, 'verb')
44     else:
45         raise ParserError("Expected a verb next.")
46
47
48 def parse_object(word_list):
49     skip(word_list, 'stop')
50     next = peek(word_list)
51
52     if next == 'noun':
53         return match(word_list, 'noun')
54     if next == 'direction':
55         return match(word_list, 'direction')
56     else:
57         raise ParserError("Expected a noun or direction next.")
58
59
60 def parse_subject(word_list, subj):
```

```
61     verb = parse_verb(word_list)
62     obj = parse_object(word_list)
63
64     return Sentence(subj, verb, obj)
65
66
67 def parse_sentence(word_list):
68     skip(word_list, 'stop')
69
70     start = peek(word_list)
71
72     if start == 'noun':
73         subj = match(word_list, 'noun')
74         return parse_subject(word_list, subj)
75     elif start == 'verb':
76         # assume the subject is the player then
77         return parse_subject(word_list, ('noun', 'player'))
78     else:
79         raise ParserError("Must start with subject, object, or verb
not: %s" % start)
```


关于异常

你已经简单学过关于异常的一些内容，但还没学过怎样引发它们。这个习题的代码演示了如何用前面定义的`ParserError`来引发异常。注意`ParserError`是一个定义为`Exception`类型的类。另外要注意我们是怎样使用`raise`这个关键字来引发异常的。

你的测试代码应该也要测试到这些异常，这个我也会演示给你如何实现。

应该测试的东西

为习题 49 写一个完整的测试方案，确认代码中所有的东西都能正常工作。将这些测试放到`tests/parser_tests.py`中，与上一个习题类似。其中包括异常测试——输入一个错误的句子它会抛出一个异常来。

使用`assert_raises`函数来检查异常，在`nose`文档里查看相关的内容，学着用它写针对“执行失败”的测试，这也是测试很重要的一个方面。从`nose`文档中学会使用 `assert_raises`以及其他一些函数。

写完测试以后，你应该就明白这段程序的工作原理了，而且也学会了如何为别人的程序写测试代码。相信我，这是一种非常有用的技能。

附加练习

- 1.修改parse_方法，将它们放到一个类里边，而不仅仅是只作为一个方法。这两种程序设计你喜欢哪一种呢？
- 2.提高parser对错误输入的抵御能力，这样即使用户输入了你预定义语汇之外的单词，你的程序也能正常运行。
- 3.改进文法，让它可以处理更多的东西，如数字。
- 4.想想在游戏里你可以使用这个语句类对用户输入做哪些有趣的事情。

常见问题回答

assert_raises老是弄不对。

确认你写的是 `assert_raises(exception, callable, parameters)`而不是 `assert_raises (exception, callable(parameters))`。注意第二个格式所做的其实是调用这个函数，并将函数的返回值作为参数传给**assert_raises**，这样做是错误的。必须把要调用的函数和它的参数分别传入**assert_raises**中。

习题50 你的第一个网站

在这个习题以及后面的习题中，你的任务是把前面创建的游戏做成网页版。这是本书的最后三个习题，这些内容对你来说难度会相当大，你要花些时间才能做出来。在开始这个习题以前，你必须已经成功地完成了习题46的内容，正确安装了pip，而且学会了如何安装软件包以及如何创建骨架项目目录。如果不记得这些内容，就回到习题46复习一遍。

安装lpthw.web

在创建你的第一个 Web 应用程序之前，你需要安装一个“Web 框架”，它的名字叫lpthw.web。所谓的“框架”通常是指“让某件事情做起来更容易的软件包”。在Web应用程序的世界里，人们创建了各种各样的“Web框架”，用来解决他们在搭建网站时遇到的问题，然后把这些解决方案用软件包的方式发布出来，这样你就可以下载这些软件包，用它们引导你自己的项目了。

可选的框架有很多很多，不过在这里我们将使用lpthw.web框架。你可以先学会它，等到差不多的时候再去接触其他框架。lpthw.web本身挺不错的，就算你一直使用也没关系。

使用pip安装lpthw.web:

```
$ sudo pip install lpthw.web
```

```
[sudo] password for zedshaw:
```

```
Downloading/unpacking lpthw.web
```

```
Running setup.py egg_info for package lpthw.web
```

```
Installing collected packages: lpthw.web
```

```
Running setup.py install for lpthw.web
```

```
Successfully installed lpthw.web
```

```
Cleaning up...
```

上面是Linux和Mac OS X系统下的安装命令，如果你使用的是Windows，只要把sudo去掉就可以了。如果无法正常安装，请回到习题46，确认自己学会了里边的内容。

警告 其他Python程序员会警告你说lpthw.web只是另外一个叫做

web.py的Web框架的代码fork，而web.py里面又包含了太多的“魔法”。如果他们这么说，你告诉他们Google App Engine最早用的就是web.py，但没有一个Python程序员抱怨过它里边包含了太多的“魔法”，因为Google用它也没啥问题。如果Google觉得它可以，那它对你来说也不会差。所以还是继续学习吧，他们这些说法与其说是教导你，不如说是拿他们自己的教条束缚你，还是忽略这些说法好了。

写一个简单的“Hello World”项目

现在做一个非常简单的“Hello World”项目出来。首先你要创建一个项目目录：

```
$ cd projects
```

```
$ mkdir gothonweb
```

```
$ cd gothonweb
```

```
$ mkdir bin gothonweb tests docs templates
```

```
$ touch gothonweb/__init__.py
```

```
$ touch tests/__init__.py
```

你最终的目的是把习题 42 中的游戏做成一个 Web 应用程序，所以你的项目名称叫做gothonweb。不过在此之前，你需要创建一个最基本的lpthw.web应用程序，将下面的代码放到bin/app.py中。

ex50.py

```
1  import web
2
3  urls = (
4      '/', 'index'
5  )
6
7  app = web.application(urls, globals())
8
9  class index:
10     def GET(self):
```



```
11         greeting = "Hello World"
12         return greeting
13
14 if __name__ == "__main__":
15     app.run()
```

然后使用下面的方法来运行这个Web应用程序：

```
$ python bin/app.py
```

```
http://0.0.0.0:8080/
```

不过，如果你执行下面的命令你就错了：

```
$ cd bin/          # WRONG! WRONG! WRONG!
```

```
$ python app.py    # WRONG! WRONG! WRONG!
```

在所有的Python项目中，你都不需要进到底层目录去运行东西。你应该待在最上层目录运行，这样才能保证所有的模块和文件都能被正常访问到。如果你犯了这个错误，请回到习题46学习一下关于项目布局的知识。

最后，使用你的Web浏览器打开<http://localhost:8080/>，你应该看到两样东西，首先是浏览器里显示了Hello, World!，然后是你的命令行终端显示了如下输出：

```
$ python bin/app.py
```

```
http://0.0.0.0:8080/
```

```
127.0.0.1:59542 - - [13/Jun/2011 11:44:43] "http/1.1 GET /" - 200 OK
```

```
127.0.0.1:59542 - - [13/Jun/2011 11:44:43] "http/1.1 GET /favicon.ico"
- 404 Not Found
```

这些是lpthw.web打印出的日志（log）信息，从这些信息可以看出服务器在运行，而且能了解到程序在浏览器背后做了些什么事情。这些信息还有助于你发现程序的问题。例如，在最后一行它告诉你浏览器试图获取/favicon.ico，但是这个文件并不存在，因此它返回的状态码是

404 Not Found。

到这里，我还没有讲到任何Web相关的工作原理，因为首先你需要完成准备工作，以便后面的学习能顺利进行，接下来的两个习题中会有详细的解释。我会要求你用各种方法把你的lpthw.web 应用程序弄坏，然后再将其重新构建起来，这样做的目的是让你明白运行lpthw.web程序需要准备好哪些东西。

会发生什么

在浏览器访问你的Web应用程序时，发生了下面这些事情。

1.浏览器通过网络连接到你的电脑，它的名字叫做 `localhost`，这是一个标准称谓，表示的就是网络中你的这台计算机，不管它实际名字是什么，你都可以使用`localhost`来访问。它使用的网络端口是8080。

2.连接成功以后，浏览器对`bin/app.py`这个应用程序发出了HTTP请求（request），要求访问URL/，这通常是一个网站的第一个URL。

3.在`bin/app.py`里，有一个列表，里边包含了URL和类的匹配关系。这里只定义了一组匹配，那就是`('/', 'index')`的匹配。它的含义是：如果有人使用浏览器访问/这一级目录，`lpthw.web` 将找到并加载 `class index`，从而用它处理这个浏览器请求。

4.现在 `lpthw.web` 找到了 `class index`，然后针对这个类的一个实例调用了`index.GET` 这个方法。该函数运行后返回了一个字符串，以供 `lpthw.web` 将其传递给浏览器。

5.最后，`lpthw.web`完成了对于浏览器请求的处理，将响应（response）回传给浏览器，于是你就看到了现在的页面。

确定你真的弄懂了这些，你需要画一个示意图，来理清信息是如何从浏览器传递到`lpthw.web`，再到`index.GET`，再回到你的浏览器的。

修正错误

第一步，把第11行的`greeting`变量赋值删掉，然后刷新浏览器。你应该会看到一个错误页面，你可以通过这一页丰富的错误信息看出你的程序崩溃的原因是什么。当然你已经知道出错的原因是`greeting`的赋值丢失了，不过`lpthw.web`还是会给你一个挺好的错误页面，让你能找到出错的具体位置。试试在这个错误页面上做以下操作。

- 1.检查每一段Local vars输出（用鼠标点击它们），追踪里边提到的变量名称，以及它们是在哪些代码文件中用到的。

- 2.阅读Request Information一节，看看里边哪些知识是你已经熟悉的。请求是浏览器发给你的`gothonweb`应用程序的信息。这些知识对于日常Web浏览没有什么用处，但现在你要学会这些，以便写出Web应用程序来。

- 3.试着把这个小程序的别的位置改错，探索一下会发生什么事情。`lpthw.web` 会把一些错误信息和栈跟踪（stack trace）信息显示在终端上，所以别忘了检查终端的信息输出。

创建基本的模板文件

你已经试过用各种方法把这个lpthw.web程序改错，不过你有没有注意到“Hello World”并不是一个好HTML网页呢？这是一个Web应用程序，所以需要有一个合适的HTML响应页面才对。为了达到这个目的，下一步你要做的是将“Hello World”以较大的绿色字体显示出来。

第一步是创建一个templates/index.html文件，内容如下。

index.html

```
$def with (greeting)
<html>
  <head>
    <title>Gothons Of Planet Percal #25</title>
  </head>
  <body>
    $if greeting:
      I just wanted to say <em style="color: green; font-size:
2em;">$greeting
    </em>.
    $else:
      <em>Hello</em>, world!
  </body>
</html>
```

如果你学过HTML，这些内容看上去应该很熟悉。如果你没学过HTML，那应该去研究一下，试着用HTML写几个网页，以便了解它的

工作原理。不过我们这里的 `HTML` 文件其实是一个“模板”（`template`），如果你向模板提供一些参数，`lpthw.web`就会在模板中找到对应的位置，将参数的内容填充到模板中。例如，每一个出现 `$greeting` 的位置都是传递给模版的变量，这些变量会改变模板显示的内容。

为了让`bin/app.py`处理模板，需要写一写代码，告诉`lpthw.web`到哪里去找到模板进行加载，以及如何渲染（`render`）这个模板，按下面的方式修改你的`app.py`。

`app.py`

```
1  import web
2
3  urls = (
4      '/', 'Index'
5  )
6
7  app = web.application(urls, globals())
8
9  render = web.template.render('templates/')
10
11  class Index(object):
12      def GET(self):
13          greeting = "Hello World"
14          return render.index(greeting = greeting)
15
16  if __name__ == "__main__":
17      app.run()
```

特别注意一下`render`这个新变量名，注意我修改了`index.GET`的最后

一行，让它返回了`render.index()`，并且将`greeting`变量作为参数传递给了这个函数。

改好上面的代码后，刷新一下浏览器中的网页，你应该会看到一条和之前不同的绿色信息输出。你还可以在浏览器中通过“查看源文件”（View Source）看到模板被渲染成了标准有效的HTML源代码。

这么讲也许有些太快了，我来详细解释一下模板的工作原理吧。

1.在 `bin/app.py` 里面添加了一个叫做 `render` 的新变量，它本身是一个`web.template.render`对象。

2.将`templates/`作为参数传递给了这个对象，这样就让`render`知道了从哪里去加载模板文件。

3.在后面的代码中，当浏览器一如既往地触发了`index.GET`以后，它没有再返回简单的`greeting`字符串，取而代之的是调用了`render.index`，而且将问候语句作为一个变量传递给它。

4.这个 `render_template` 函数可以说是一个“魔法函数”，它看到了你需要的是`index.html`，于是就跑到`templates/`目录下找到名字为`index.html`的文件，然后就把它渲染一遍（叫“转换一遍”也可以）。

5.在 `templates/index.html` 文件中，可以看到初始定义一行中说这个模板需要使用一个叫`greeting`的参数，这和函数定义中的格式差不多。另外和Python语法一样，模板文件是缩进敏感的，所以要确认自己弄对了缩进。

6.最后，让`templates/index.html`去检查`greeting`这个变量，如果这个变量存在，就打印出变量的内容；如果不存在，就会打印出一个默认的问候信息。

要深入理解这个过程，你可以修改`greeting`变量以及HTML模板的内容，看看会有什么效果。然后创建一个叫做 `templates/foo.html` 的模板，并且使用一个新的 `render.foo()`（而不是像之前一样使用 `render.index()`）去渲染它。从这个过程也可以看出，`render`调用的函数名称只要跟

templates/下的.html文件名匹配到，这个HTML模板就可以被渲染了。

附加练习

- 1.到<http://webpy.org/>阅读里边的文档，它其实和lpthw.web是同一个项目。
- 2.实验一下你在上述网站看到的所有内容，包括里边的代码示例。
- 3.阅读一下与HTML5和CSS3相关的东西，自己练习着写几个.html和.css文件。
- 4.如果有懂Django朋友可以帮你，你可以试着使用Django完成一下习题50、习题51和习题52，看看结果会是什么样子的。

常见问题回答

我没法连接到`http://localhost:8080/`。

那就试试`http://127.0.0.1:8080/`。

lpthw.web和**web.py**有什么不同？

一样的。我只不过“锁定”了**web.py**的某个版本，把它命名为**lpthw.web**，这样大家用的版本就都是一样的了。就算日后**web.py**升级升得面目全非，我也无需更新本书的内容。

我找不到**index.html**（或者别的文件）。

很有可能是你先跑了 `cd bin/`然后才开始做项目的。不要这么做，所有的指令都应该在**bin/**的上一层目录中完成，所以如果你无法运行 `python bin/app.py`，就说明你不在正确的目录下。

为什么调用模板时要写**greeting=greeting**？

这一句并不是赋值给**greeting**，而是将一个命名参数传到模板中。这也算是一种赋值，不过只会在模板函数的调用中生效。

我的计算机的端口**8080**无法使用。

也许是哪个杀毒软件占用了这个端口，那就换一个端口好了。

安装**lpthw.web**时出现**ImportError "No module named web"**。

很有可能是你在系统中安装了多个版本的Python，而在这里你用了错误的一个，或者由于**pip** 版本太旧导致安装没有正确完成。试着卸载并重装 **lpthw.web**。如果还不行，那就再仔细检查一下，确认自己用了正确版本的Python。

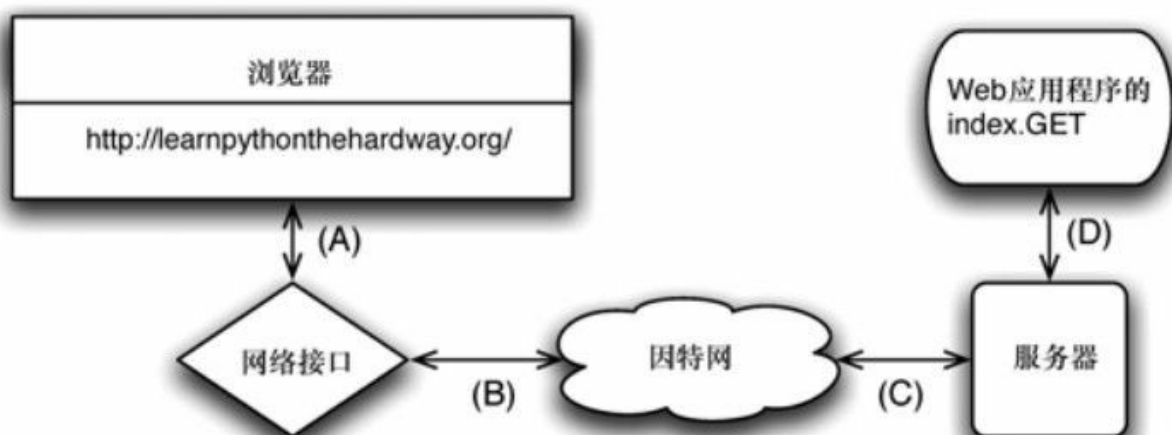
习题51 从浏览器中获取输入

虽然能让浏览器显示“Hello World”是很有趣的一件事情，但是如果能让用户通过表单（**form**）向应用程序提交文本就更有意思了。在这个习题中，我们将使用表单改进你的Web应用程序，并且将用户相关的信息保存到他们的“会话”（**session**）中。

Web的工作原理

该学点无趣的东西了。在创建表单前需要先多学一点关于Web的工作原理。这里讲的并不完整，但是相当准确，在你的程序出错时，它会帮你找到出错的原因。另外，如果你理解了表单的应用，那么创建表单对你来说就会更容易了。

我将以一张简单的图讲起，它向你展示了Web请求的各个不同的部分，以及信息传递的大致流程。



为了方便讲述HTTP请求（request）的流程，我在每条线上加了字母标签以示区别。

1.你在浏览器中输入网址<http://learnpythonthehardway.org/>，然后浏览器会通过你的计算机的网络设备发出请求（线路A）。

2.你的请求被传送到互联网（线路B），然后再抵达远程服务器（线路C），然后我的服务器将接受这个请求。

3.服务器接受请求后，我的Web应用程序就去处理这个请求（线路D），然后我的Python代码就会去运行index.GET这个“处理程

序”（**handler**）。

4.在代码返回的时候，我的Python服务器就会发出响应，这个响应会再通过线路D传递到你的浏览器。

5.这个网站所在的服务器将获取由线路D发出的响应，然后通过线路C传至因特网。

6.响应通过互联网由线路 B 传至你的计算机，计算机的网卡再通过线路 A 将响应传给你的浏览器。

7.最后，你的浏览器显示了这个响应的内容。

这段详解中用到了一些术语，你需要掌握这些术语，以便在谈论Web应用程序时你能明白而且应用它们。

浏览器（**browser**）。这是你几乎每天都会用到的软件。大部分人不知道它真正的原理，只会把它叫作“网”。它的作用其实是接收你输入到地址栏的网址（如<http://learnpythonthehardway.org>），然后使用该信息向该网址对应的服务器提出请求。

地址（**address**）。通常这是一个像<http://learnpythonthehardway.org/>一样的URL（Uniform Resource Locator，统一资源定位器），它告诉浏览器该打开哪个网站。前面的http指出了你要使用的协议，这里用的是“超文本传输协议”（Hyper-Text Transport Protocol，HTTP）。你还可以试试 <ftp://ibiblio.org/>，这是一个“文件传输协议”（File Transport Protocol，FTP）的例子。learnpythonthehardway.org 这部分是“主机名”（hostname），也就是一个便于人阅读和记忆的地址，主机名会被匹配到一串叫作“IP地址”的数字上面，这个“IP 地址”就相当于网络中一台计算机的电话号码，通过这个号码可以访问这台计算机。最后，URL中还可以跟随一个“路径”，例如<http://learnpythonthehardway.org/book/>中的/book/，它对应的是服务器上的某个文件或者某些资源，通过访问这样的网址，你可以向服务器发出请求，然后获得这些资源。网站地址还有很多别的组成部分，不过上面讲的这些是最主要的。

连接（**connection**）。一旦浏览器知道了你用的协议（**http**）、你想联络的服务器（**learnpythonthehardway.org**）及要从服务器上获得的资源，它就要去创建一个连接。这个过程中，浏览器让操作系统

（**Operating System, OS**）打开计算机的一个“端口”（**port**）（通常是80端口），端口准备好以后，操作系统会回传给你的程序一个类似文件的东西，它所做的事情就是通过网络传输和接收数据，让你的计算机和**learnpythonthehardway.org**这个网站所属的服务器之间实现数据交流。当你使用**http://localhost:8080/**访问自己的站点时，发生的事情其实是一样的，只不过这次你告诉了浏览器要访问的是你自己的计算机

（**localhost**），要使用的端口不是默认的80，而是8080。你还可以直接访问**http://learnpythonthehardway.org:80/**，这和不输入端口效果一样，因为**HTTP**的默认端口本来就是80。

请求（**request**）。浏览器通过你提供的地址建立了连接，现在它需要从远端服务器要到它（或你）想要的资源。如果在**URL**的结尾加了**/book/**，那你想要的就是**/book/**对应的文件或资源，大部分的服务器会直接为你调用**/book/index.html**这个文件，不过我们就假装不存在好了。浏览器为了获得服务器上的资源，它需要向服务器发送一个“请求”。这里就不讲细节了，为了得到服务器上的内容，你必须先向服务器发送一个请求才行。有意思的是，“资源”不一定非要是文件。例如，当浏览器向你的应用程序提出请求的时候，服务器返回的其实是你的**Python**代码生成的一些东西。

服务器（**server**）。服务器指的是浏览器另一端连接的计算机，它知道如何回应浏览器请求的文件和资源。大部分的**Web**服务器只要发送文件就可以了，这也是服务器流量的主要部分。不过你学的是使用**Python**构建一个服务器，这个服务器知道如何接受请求，然后返回用**Python**处理过的字符串。当使用这种处理方式时，其实是假装把文件发给了浏览器，而你用的都只是代码而已。就像你在习题50中看到的，要

构建一个“响应”其实也不需要多少代码。

响应（**response**）。这就是你的服务器回复你的请求而发回至浏览器的 HTML，它里边可能有CSS、JavaScript或者图像等内容。以文件响应为例，服务器只要从磁盘读取文件，发送给浏览器就可以了，不过它还要将这些内容包在一个特别定义的“首部信息”（**header**）中，这样浏览器就会知道它获取的是什么类型的内容。以你的Web应用程序为例，你发送的其实还是一样的东西，包括首部信息也一样，只不过这些数据是你用Python代码即时生成的。

这可以算是你能在网上找到的关于浏览器如何访问网站的最快的快速课程了。这个习题可以帮你更容易地理解这个习题，如果你还是不明白，就到处找资料，多多了解这方面的信息，直到弄明白为止。有一个很好的方法就是，你对照着上面的图，将你在习题50中创建的Web应用程序的内容分成几个部分，让其中的各部分对应到上面的图。如果你能正确地将程序的各部分对应到这张图，你就大致开始明白它的工作原理了。

表单的工作原理

熟悉“表单”最好的方法就是写一个可以接收表单数据的程序出来，然后看你可以对它做些什么。先将你的bin/app.py修改成下面的样子。

form_test.py

```
1  import web
2
3  urls = (
4      '/hello', 'Index'
5  )
6
7
8  app = web.application(urls, globals())
9
10 render = web.template.render('templates/')
11
12 class Index(object):
13     def GET(self):
14         form = web.input(name="Nobody")
15         greeting = "Hello, %s" % form.name
16
17         return render.index(greeting = greeting)
18
19 if __name__ == "__main__":
```


重启你的Web应用程序（按Ctrl+C后重新运行），确认它已运行起来，然后使用浏览器访问`http://localhost:8080/hello`，这时浏览器应该会显示“I just wanted to say Hello, Nobody.”，接下来，将浏览器的地址改成`http://localhost:8080/hello?name=Frank`，然后你可以看到页面显示为“Hello, Frank.”，最后将`name=Frank`修改为你自己的名字，你就可以看到它对你说“Hello”了。

让我们研究一下你的程序里做过的修改。

1.这里没有直接为`greeting`赋值，而是使用了`web.input`从浏览器获取数据。这个函数会将一组“键=值”的表述作为默认参数，解析你提供的URL中的`?name=Frank`部分，然后返回一个对象，你可以通过这个对象方便地访问到表单的值。

2.然后通过`form`对象的`form.name`属性为`greeting`赋值，这句你应该已经熟悉了。3.其他的内容和以前是一样的，就不再分析了。

URL中该还可以包含多个参数。将本例的URL改成`http://localhost:8080/hello?name=Frank&greet=Hola`。然后修改代码，让它去获取`form.name`和`form.greet`，如下所示：

```
greeting = "%s, %s" % (form.greet, form.name)
```

修改完毕后，试着访问新的URL，然后将`&greet=Hola`部分删除，看看会得到什么样的错误信息。由于在`web.input(name="Nobody")`中没有为`greet`设定默认值，这样`greet`就变成了一个必需的参数，如果没有这个参数程序就会报错。现在修改一下你的程序，在`web.input`中为`greet`设一个默认值试试看。另外，你还可以设`greet=None`，这样可以通过程序检查`greet`的值是否存在，然后提供一个比较好的错误信息出来。例如：

```
form = web.input(name="Nobody", greet=None)
if form.greet:
    greeting = "%s, %s" % (form.greet, form.name)
```

```
    return render.index(greeting = greeting)
else:
    return "ERROR: greet is required."
```

创建HTML表单

在 URL 上传递参数是可以的，不过这样看上去有些丑陋，而且不方便普通人使用，你真正需要的是一个“POST表单”，这是一种包含了<form>标签的特殊HTML文件。这种表单收集用户输入并将其传递给你的Web应用程序，这和你上面实现的目的基本是一样的。

让我们来快速创建一个，从中你可以看出它的工作原理。你需要创建一个新的HTML文件，叫做templates/hello_form.html:

hello_form.html

```
1  <html>
2      <head>
3          <title>Sample Web Form</title>
4      </head>
5  <body>
6
7  <h1>Fill Out This Form</h1>
8
9  <form action="/hello" method="POST">
10      A Greeting: <input type="text" name="greet">
11      <br/>
12      Your Name: <input type="text" name="name">
13      <br/>
14      <input type="submit">
15  </form>
```

16

17 </body>

18 </html>

然后将bin/app.py改成下面这个样子。

post_form.py

1 import web

2

3 urls = (

4 '/hello', 'Index'

5)

6

7 app = web.application(urls, globals())

8

9 render = web.template.render('templates/')

10

11 class Index(object):

12 def GET(self):

13 return render.hello_form()

14

15 def POST(self):

16 form = web.input(name="Nobody", greet="Hello")

17 greeting = "%s, %s" % (form.greet, form.name)

18 return render.index(greeting = greeting)

19

20 if __name__ == "__main__":

21 app.run()

都写好以后，重启Web应用程序，然后通过浏览器访问它。

这回你会看到一个表单，它要求你输入“一个问候语句”（A Greeting）和“你的名字”（Your Name），等你输入完后点击“提交”（Submit）按钮，它就会输出一个正常的问候页面，不过这一次你的URL还是http://localhost:8080/hello，并没有包含你提交的参数。

在hello_form.html里面关键的一行是<form action="/hello" method="POST">，它告诉你的浏览器以下内容。

- 1.从表单中的各个栏位收集用户输入的数据。
- 2.让浏览器使用一种POST类型的请求，将这些数据发送给服务器。这是另外一种浏览器请求，它会将表单栏位“隐藏”起来。
- 3.将这个请求发送至/hello URL，这是由action="/hello"告诉浏览器的。

你可以看到两段<input>标签的名字（name）属性和代码中的变量是对应的，另外我们在class index中使用的不再只是GET方法，而是另一个POST方法。这个新程序的工作原理如下。

- 1.浏览器访问 Web 应用程序的/hello 目录，它发送了一个 GET 请求，于是我们的index.GET函数就运行并返回了hello_form。
- 2.你填好了浏览器的表单，然后浏览器依照<form>中的要求，将数据通过 POST 请求的方式发给Web应用程序。
- 3.Web应用程序运行了index.POST方法（而不是index.GET方法）来处理这个请求。
- 4.这个index.POST方法完成了它正常的功能，将hello页面返回，这里并没有新的东西，只是一个新函数名称而已。

作为练习，在templates/index.html中添加一个链接，让它指向/hello，这样你可以反复填写并提交表单查看结果。确认你可以解释清楚这个链接的工作原理，以及它是如何让你实现在templates/index.html和templates/hello_form.html之间循环跳转的，还有就是要明白你新修改过的Python代码，清楚在什么情况下会运行到哪一部

分代码。

创建布局模板

在下一个习题中创建游戏的过程中，你需要创建很多的小HTML页面。如果你每次都写一个完整的网页，你会很快感觉到厌烦。幸运的是，你可以创建一个“布局模板”（layout template），也就是一种提供了通用的头文件和脚注的外壳模板，你可以用它将你所有的其他网页包裹起来。好程序员会尽可能减少重复动作，所以要做一个好程序员，使用布局模板是很重要的。

将templates/index.html修改成下面这样。

index_laid_out.html

```
$def with (greeting)
```

```
$if greeting:
```

```
    I just wanted to say <em style="color: green; font-size:
2em;">$greeting</em>.
```

```
$else:
```

```
    <em>Hello</em>, world!
```

然后把templates/hello_form.html修改成下面这样。

hello_form_laid_out.html

```
<h1>Fill Out This Form</h1>
```

```
<form action="/hello" method="POST">
```

```
    A Greeting: <input type="text" name="greet">
```

```
    <br/>
```

```
    Your Name: <input type="text" name="name">
```

```
    <br/>
```

```
<input type="submit">
</form>
```

上面这些修改的目的是将每一个页面顶部和底部的反复用到的“样板代码”代码剥掉。这些被剥掉的代码会被放到一个单独的 `templates/layout.html` 文件中，从此以后，这些反复用到的代码就由 `templates/layout.html` 来提供了。

上面的都改好以后，创建一个 `templates/layout.html` 文件，具体内容如下。

`layout.html`

```
$def with (content)
<html>
<head>
    <title>Gothons From Planet Percal #25</title>
</head>
<body>
    $:content
</body>
</html>
```

这个文件和普通的模板文件类似，只是其他模板的内容将被传递给它，然后它会将其他模板的内容“包裹”起来。任何写在这里的内容都无需写在别的模板中了。要注意 `$:content` 的用法，这和其他模板变量有些不同。

最后一步，就是将 `render` 对象改成这样：

```
render = web.template.render('templates/', base="layout")
```

这会告诉 `lpthw.web` 去使用 `templates/layout.html` 作为其他模板的基础模板。重启你的应用程序观察一下，然后试着用各种方法修改你的布局模板，不要修改别的模板，看看输出会有什么变化。

为表单撰写自动测试代码

使用浏览器测试Web应用程序是很容易的，只要点刷新按钮就可以了。不过毕竟我们是程序员，如果可以写一些代码来测试我们的程序，为什么还要重复手动测试呢？接下来你要做的就是，为你的Web应用程序写一个小测试。这会用到在习题47中学过的一些东西，如果你不记得的话，可以回去复习一下。

为了让 Python 加载 bin/app.py 并进行测试，需要先做一点准备工作。首先创建一个bin/__init__.py空文件，这样Python就会将bin/当作一个目录了。（在习题52中会修改__init__.py，不过这是后话。）

我还为lpthw.web创建了一个简单的小函数，让你断言（assert）Web应用程序的响应，这个函数有一个很合适的名字，就叫assert_response。创建一个tests/tools.py文件，内容如下。

tools.py

```
1  from nose.tools import *
2  import re
3
4  def  assert_response(resp,  contains=None,  matches=None,
headers=None, status="200"):
5
6      assert status in resp.status, "Expected response %r not in %r"
% (status, resp.status)
7
8      if status == "200":
```

```

9             assert resp.data, "Response data is empty."
10
11         if contains:
12             assert contains in resp.data, "Response does not contain
%r" % contains
13
14         if matches:
15             reg = re.compile(matches)
16             assert reg.matches(resp.data), "Response does not match
%r" % matches
17
18         if headers:
19             assert_equal(resp.headers, headers)

```

准备好这个文件以后，你就可以为 `bin/app.py` 写自动测试代码了。
 创建一个叫 `tests/app_tests.py` 的新文件，具体内容如下。

`app_tests.py`

```

1  from nose.tools import *
2  from bin.app import app
3  from tests.tools import assert_response
4
5  def test_index():
6      # check that we get a 404 on the / URL
7      resp = app.request("/")
8      assert_response(resp, status="404")
9
10     # test our first GET request to /hello
11     resp = app.request("/hello")

```

```

12     assert_response(resp)
13
14     # make sure default values work for the form
15     resp = app.request("/hello", method="POST")
16     assert_response(resp, contains="Nobody")
17
18     # test that we get expected values
19     data = ['name': 'Zed', 'greet': 'Hola']
20     resp = app.request("/hello", method="POST", data=data)
21     assert_response(resp, contains="Zed")

```

最后，使用`nosetests`运行这个测试脚本，测试你的Web应用程序。

```
$ nosetests
```

```
.
```

```
-----
Ran 1 test in 0.059s
```

```
OK
```

这里我所做的就是将`bin/app.py`这个模块中的整个Web应用程序都导入进来，然后手动运行这个Web应用程序。`lpthw.web`有一个非常简单的API用来处理请求，看上去大致是下面这个样子：

```

app.request(localpart='/',          method='GET',          data=None,
host='0.0.0.0:8080',
            headers=None, https=False)

```

你可以将URL作为第一个参数，然后修改`request`的方法、`form`的数据及`header`的内容，这样，无须启动Web服务器就可以使用自动测试来测试你的Web应用程序了。

为了验证函数的响应，你需要使用`tests.tools`中定义的`assert_response`函数，用法如下：

```
assert_response(resp, contains=None, matches=None, headers=None,
status="200")
```

把调用 `app.request` 得到的响应传递给这个函数，然后将要检查的内容作为参数传递给这个函数。你可以使用 `contains` 参数来检查响应中是否包含指定的值，使用 `status` 参数可以检查指定的响应状态。这个小函数其实包含了很多的信息，所以你还是自己研究一下比较好。

在 `tests/app_tests.py` 自动测试脚本中，我首先确认/返回了一个 404 Not Found 响应，因为这个 URL 其实是不存在的。然后我检查了 `/hello` 在 GET 和 POST 两种请求的情况下都能正常工作。就算你没弄明白测试的原理，这些测试代码应该是很好读懂的。

花一些时间研究一下这个最新版的 Web 应用程序，重点研究一下自动测试的工作原理。确认你理解了将 `bin/app.py` 作为一个模块导入然后进行自动测试的流程。这是一个很重要的技巧，它会引导你学到更多东西。

附加练习

1.阅读与HTML相关的更多资料，为你的表单设计一个更好的输出格式。你可以先在纸上设计出来，然后用HTML去实现它。

2.这是一道难题，试着研究一下如何进行文件上传，通过网页上传一张图像，然后将其保存到磁盘中。

3.更难的难题，找到HTTP RFC文件（讲述HTTP工作原理的技术文件），然后尽力阅读一下。这是一篇很无趣的文档，不过偶尔你会用到里边的一些知识。

4.又是一道难题，找人帮你设置一个Web服务器，如Apache、Nginx或者thttpd。试着让服务器伺候一下你创建的.html和.css文件。如果失败了也没关系，Web服务器本来就有点儿让人失望。

5.完成上面的任务后休息一下，然后试着多创建一些 Web 应用程序。你应该仔细阅读web.py（它和 lpthw.web 是一样的）中关于会话（session）的内容，这样你就能明白如何保持用户的状态信息。

常见问题回答

我看到了**ImportError "No module named bin.app"**。

再次说明，要么是你引用的路径不对，要么是没有创建bin/___init__.py 文件，要么是在shell中没有设置PYTHONPATH=。。记住这些解决方案，这些问题会经常遇到，到处问人解决方案只会拖慢你的速度。

运行模板时发生**__template__() takes no arguments (1 given)**错误。

你很可能忘记了在模板开头放置\$def with (greeting)或者类似的变量声明。

习题52 创建Web游戏

这本书马上就要结束了。这个习题对你将是一个真正的挑战。完成这个习题之后，你就可以算是一个能力相当不错的Python初学者了。虽然还需要多读一些书，多写一些程序，不过你已经具备进一步学习的功底了。接下来的学习就只是时间、动力及资源的问题了。

在这个习题中，我们不会去创建一个完整的游戏，相反，我们会为习题42中的游戏创建一个“引擎”（engine），让这个游戏能够在浏览器中运行起来。这会涉及重构习题 42 中的游戏，混合习题 47 中的结构，添加自动测试代码，最后创建一个可以运行这个游戏的Web引擎。

这是一个很庞大的习题。预计你要花一周到一个月才能完成。最好的方法是一点一点来，每晚完成一点，在进行下一步之前确认上一步已经正确完成。

重构习题43中的游戏

你已经在两个习题中修改了gothonweb项目，这个习题中会再修改一次。你学习的这种修改的技术叫做“重构”，或者用我喜欢的讲法来说，叫“修理”。重构是一个编程术语，它指的是清理旧代码或者为旧代码添加新功能的过程。你其实已经做过这样的事情了，只不过不知道这个术语而已。重构是软件开发中经历的最习以为常的事情。

在这个习题中你要做的是将习题47中的可以测试的房间地图和习题43中的游戏这两样东西合并到一起，创建一个新的游戏结构。游戏的内容不会发生变化，只不过我们会通过“重构”让它有一个更好的结构而已。

第一步是将 `ex47/game.py` 的内容复制到 `gothonweb/map.py` 中，然后将 `tests/ex47_tests.py` 的内容复制到 `tests/map_tests.py` 中，然后再次运行 `nosetests`，确认它们还能正常工作。

注意 从现在开始，我不会再展示运行测试的输出，我假设你会回去运行这些测试，而且知道什么样的输出是正确的。

将习题47的代码复制完毕后，就该开始重构它，让它包含习题43中的地图。我一开始会把基本结构为你准备好，然后你需要去完成 `map.py` 和 `map_tests.py` 里边的内容。

首先要做的是用 `Room` 这个类来构建地图的基本结构。

`map.py`

```
1 class Room(object):
2
3     def __init__(self, name, description):
```



```
4         self.name = name
5         self.description = description
6         self.paths = []
7
8     def go(self, direction):
9         return self.paths.get(direction, None)
10
11    def add_paths(self, paths):
12        self.paths.update(paths)
13
14
15    central_corridor = Room("Central Corridor",
16    """
17    The Gothons of Planet Percal #25 have invaded your ship and
destroyed
18    your entire crew.  You are the last surviving member and your last
19    mission is to get the neutron destruct bomb from the Weapons
Armory,
20    put it in the bridge, and blow the ship up after getting into an
21    escape pod.
22
23    You're running down the central corridor to the Weapons Armory
when
24    a Gothon jumps out, red scaly skin, dark grimy teeth, and evil
clown costume
25    flowing around his hate filled body.  He's blocking the door to the
26    Armory and about to pull a weapon to blast you.
```

```
27  """)
28
29
30  laser_weapon_armory = Room("Laser Weapon Armory",
31  """)
32  Lucky for you they made you learn Gothon insults in the academy.
33  You tell the one Gothon joke you know:
34  Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr, fur fvgf
nebhaq gur ubhfr.
35  The Gothon stops, tries not to laugh, then busts out laughing and
can't move.
36  While he's laughing you run up and shoot him square in the head
37  putting him down, then jump through the Weapon Armory door.
38
39  You do a dive roll into the Weapon Armory, crouch and scan the
room
40  for more Gothons that might be hiding.  It's dead quiet, too quiet.
41  You stand up and run to the far side of the room and find the
42  neutron bomb in its container.  There's a keypad lock on the box
43  and you need the code to get the bomb out.  If you get the code
44  wrong 10 times then the lock closes forever and you can't
45  get the bomb.  The code is 3 digits.
46  """)
47
48
49  the_bridge = Room("The Bridge",
50  """)
```

51 The container clicks open and the seal breaks, letting gas out.
52 You grab the neutron bomb and run as fast as you can to the
53 bridge where you must place it in the right spot.

54

55 You burst onto the Bridge with the neutron destruct bomb
56 under your arm and surprise 5 Goths who are trying to
57 take control of the ship. Each of them has an even uglier
58 clown costume than the last. They haven't pulled their
59 weapons out yet, as they see the active bomb under your
60 arm and don't want to set it off.

61 """)

62

63

64 escape_pod = Room("Escape Pod",

65 """)

66 You point your blaster at the bomb under your arm

67 and the Goths put their hands up and start to sweat.

68 You inch backward to the door, open it, and then carefully
69 place the bomb on the floor, pointing your blaster at it.

70 You then jump back through the door, punch the close button
71 and blast the lock so the Goths can't get out.

72 Now that the bomb is placed you run to the escape pod to
73 get off this tin can.

74

75 You rush through the ship desperately trying to make it to
76 the escape pod before the whole ship explodes. It seems like
77 hardly any Goths are on the ship, so your run is clear of

```
78  interference.  You get to the chamber with the escape pods, and
79  now need to pick one to take.  Some of them could be damaged
80  but you don't have time to look.  There's 5 pods, which one
81  do you take?
82  """)
83
84
85  the_end_winner = Room("The End",
86  """)
87  You jump into pod 2 and hit the eject button.
88  The pod easily slides out into space heading to
89  the planet below.  As it flies to the planet, you look
90  back and see your ship implode then explode like a
91  bright star, taking out the Gothon ship at the same
92  time.  You won!
93  """)
94
95
96  the_end_loser = Room("The End",
97  """)
98  You jump into a random pod and hit the eject button.
99  The pod escapes out into the void of space, then
100 implodes as the hull ruptures, crushing your body
101 into jam jelly.
102 """)
103 )
104
```

```
105 escape_pod.add_paths([
106     '2': the_end_winner,
107     '*': the_end_loser
108 ])
109
110 generic_death = Room("death", "You died.")
111
112 the_bridge.add_paths([
113     'throw the bomb': generic_death,
114     'slowly place the bomb': escape_pod
115 ])
116
117 laser_weapon_armory.add_paths([
118     '0132': the_bridge,
119     '*': generic_death
120 ])
121
122 central_corridor.add_paths([
123     'shoot!': generic_death,
124     'dodge!': generic_death,
125     'tell a joke': laser_weapon_armory
126 ])
127
128 START = central_corridor
```

你会发现Room类和地图有一些问题。

1.我们必须把以前放在if-else结构中的房间描述做成每个房间的一部分。这样房间的次序就不会被打乱了，这对我们的游戏是一件好事。

这是你后面要修改的东西。

2.原版游戏中我们使用了专门的代码来生成一些内容，如炸弹的激活键码、舰舱的选择等，这次我们做游戏时就先使用默认值好了，不过后面的附加练习里，我会要求你把这些功能再加入到游戏中。

3.我为游戏中所有错误决策的失败结尾写了一个`generic_death`，你需要去补全这个函数。你需要把原版游戏中所有的场景结局都加进去，并确保代码能正确运行。

4.我添加了一种新的转换模式，以`"*"`为标记，用来在游戏引擎中实现“捕获所有操作”的功能。

等把上面的代码基本写好以后，接下来就是你必须继续写的自动测试 `tests/map_test.py` 了。

`map_tests.py`

```
1  from nose.tools import *
2  from gothonweb.map import *
3
4  def test_room():
5      gold = Room("GoldRoom",
6                  """This room has gold in it you can
grab.There's a
7                      door to the north.""")
8      assert_equal(gold.name, "GoldRoom")
9      assert_equal(gold.paths, [])
10
11  def test_room_paths():
12      center = Room("Center", "Test room in the center.")
13      north = Room("North", "Test room in the north.")
14      south = Room("South", "Test room in the south.")
```

```

15
16     center.add_paths(['north': north, 'south': south])
17     assert_equal(center.go('north'), north)
18     assert_equal(center.go('south'), south)
19
20 def test_map():
21     start = Room("Start", "You can go west and down a hole.")
22     west = Room("Trees", "There are trees here, you can go
east.")
23     down = Room("Dungeon", "It's dark down here, you can go
up.")
24
25     start.add_paths(['west': west, 'down': down])
26     west.add_paths(['east': start])
27     down.add_paths(['up': start])
28
29     assert_equal(start.go('west'), west)
30     assert_equal(start.go('west').go('east'), start)
31     assert_equal(start.go('down').go('up'), start)
32
33 def test_gothon_game_map():
34     assert_equal(START.go('shoot!'), generic_death)
35     assert_equal(START.go('dodge!'), generic_death)
36
37     room = START.go('tell a joke')
38     assert_equal(room, laser_weapon_armory)

```

你在这个习题中的任务是完成地图，并且让自动测试可以完整地检

查整个地图。这包括将所有的generic_death对象修正为游戏中实际的失败结尾。让你的代码成功运行起来，并让你的测试越全面越好。后面我们会对地图做一些修改，到时候这些测试将用来确保修改后的代码还可以正常工作。

会话和用户跟踪

在Web应用程序运行的某个位置，你需要追踪一些信息，并将这些信息和用户的浏览器关联起来。在HTTP协议的框架中，Web环境是“无状态”的，这意味着你的每一次请求和你的其他请求都是相互独立的。如果你请求了页面A，输入了一些数据，然后点了一个页面B的链接，那你发送给页面A的数据就全部消失了。

解决这个问题的方法是为Web应用程序建立一个很小的数据存储，给每个浏览器进程赋予一个独一无二的数字，用来跟踪浏览器所做的事情。这个存储通常用数据库或者存储在磁盘上的文件来实现。在lpthw.web 这个小框架中实现这样的功能是很容易的，下面就是一个这样的例子。

session.sample.py

```
1  import web
2
3  web.config.debug = False
4
5  urls = (
6      "/count", "count",
7      "/reset", "reset"
8  )
9  app = web.application(urls, locals())
10 store = web.session.DiskStore('sessions')
11 session = web.session.Session(app, store, initializer=['count': 0])
```

```
12
13 class count:
14     def GET(self):
15         session.count += 1
16         return str(session.count)
17
18 class reset:
19     def GET(self):
20         session.kill()
21         return ""
22
23 if __name__ == "__main__":
24     app.run()
```

为了实现这个功能，需要创建一个 `sessions/` 文件夹作为程序的会话存储位置，创建好以后运行这个程序，然后检查 `/count` 页面，刷新一下这个页面，看计数会不会累加上去。关掉浏览器后，程序就会“忘掉”之前的位置，这也是我们的游戏所需的功能。有一种方法可以让浏览器永远记住一些信息，不过这会让测试和开发变得更难。如果你回到 `/reset` 页面，然后再访问 `/count` 页面，你可以看到你的计数器被重置了，因为你已经关掉了这个会话。

你需要花点时间弄懂这段代码，注意会话开始时 `count` 的值是如何设为 0 的，另外再看看 `sessions/` 下面的文件，看能不能打开。下面是我打开一个 Python 会话并解码的过程：

```
>>> import pickle
>>> import base64
>>> base64.b64decode(open("sessions/XXXXXX").read())
"
```

```
(dp1\nS'count'\np2\nI1\nsS'ip'\np3\nV127.0.0.1\np4\nsS'session_id'\np5\nS'X[
>>>
>>> x = base64.b64decode(open("sessions/XXXXXX").read())
>>>
>>> pickle.loads(x)
{'count': 1, 'ip': u'127.0.0.1', 'session_id': 'XXXXXX'}
```

所以，会话其实就是使用pickle和base64这些库写到磁盘上的字典。存储和管理会话的方法很多，大概和 Python 的 Web 框架那么多，所以了解它们的工作原理并不是很重要。当然如果你需要调试或者清空会话，知道点儿原理还是有用的。

创建引擎

你应该已经写好了游戏地图和它的单元测试代码。现在要你制作一个简单的游戏引擎，用来让游戏中的各个房间运转起来，从玩家收集输入，并且记住玩家所在的位置。我们将用到你刚学过的会话来制作一个简单的引擎，让它可以：

1. 为新用户启动新的游戏；
2. 将房间展示给用户；
3. 接收用户的输入；
4. 在游戏中处理用户的输入；
5. 显示游戏的结果，继续游戏，直到玩家角色死亡为止。

为了创建这个引擎，你需要将 `bin/app.py` 搬过来，创建一个功能完备的、基于会话的游戏引擎。这里的难点是，我会先使用基本的HTML文件创建一个非常简单的版本，接下来将由你完成它。基本的引擎是下面这个样子的：

`app.py`

```
1  import web
2  from gothonweb import map
3
4  urls = (
5      '/game', 'GameEngine',
6      '/', 'Index',
7  )
8
```

```

9  app = web.application(urls, globals())
10
11  # little hack so that debug mode works with sessions
12  if web.config.get('_session') is None:
13      store = web.session.DiskStore('sessions')
14      session = web.session.Session(app, store,
15                                     initializer=['room':
None])
16      web.config._session = session
17  else:
18      session = web.config._session
19
20  render = web.template.render('templates/', base="layout")
21
22
23  class Index(object):
24      def GET(self):
25          # this is used to "setup" the session with starting values
26          session.room = map.START
27          web.seeother("/game")
28
29
30  class GameEngine(object):
31
32      def GET(self):
33          if session.room:
34              return render.show_room(room=session.room)

```

```

35         else:
36             # why is there here? do you need it?
37             return render.you_died()
38
39     def POST(self):
40         form = web.input(action=None)
41
42         # there is a bug here, can you fix it?
43         if session.room and form.action:
44             session.room = session.room.go(form.action)
45
46         web.seeother("/game")
47
48 if __name__ == "__main__":
49     app.run()

```

在这个脚本里你可以看到更多的新东西，不过了不起的事情是，整个基于网页的游戏引擎只要一个小文件就可以做到了。这段脚本里最有技术含量的就是将会话带回来的那几行，这对于调试模式下的代码重载是必需的，否则每次刷新网页，会话就会消失，游戏也不会再继续了。

在运行bin/app.py之前，你需要修改PYTHONPATH环境变量。不知道什么是环境变量？要运行一个最基本的Python程序，你就得学会环境变量，用Python的人就喜欢这样：

在终端输入下面的内容：

```
export PYTHONPATH=$PYTHONPATH:.
```

如果用的是Windows，那就在PowerShell中输入以下内容：

```
$env:PYTHONPATH = "$env:PYTHONPATH;."
```

你只要针对每一个shell会话输入一次就可以了，不过如果你运行

Python代码时看到了导入错误，那就需要去执行一下上面的命令，或者是因为你上次执行的有错才导致导入错误的。

接下来需要删掉templates/hello_form.html和templates/index.html，然后重新创建上面代码中提到的两个模板。下面是一个非常简单的templates/show_room.html，供你参考。

show_room.html

```
$def with (room)
<h1> $room.name </h1>
<pre>
$room.description
</pre>
$if room.name == "death":
    <p><a href="/">Play Again?</a></p>
$else:
    <p>
        <form action="/game" method="POST">
            -    <input    type="text"    name="action">    <input
type="SUBMIT">
        </form>
    </p>
```

这就用来显示游戏中的房间的模板。接下来，你需要在用户跑到地图的边界时，用一个模板告诉用户，他的角色的死亡信息，也就是templates/you_died.html这个模板。

you_died.html

```
<h1>You Died!</h1>
<p>Looks like you bit the dust.</p>
<p><a href="/">Play Again</a></p>
```

准备好这些文件就可以做下面的事情了。

1.再次运行测试代码`tests/app_tests.py`，这样就可以测试这个游戏。由于会话的存在，你可能顶多只能实现几次点击，不过你应该可以做出一些基本的测试来。

2.删除`sessions/*`下的文件，再重新运行一遍游戏，确认游戏是从一开始运行的。

3.运行`python bin/app.py`脚本，试玩一下你的游戏。

你需要和往常一样刷新和修正你的游戏，慢慢修改游戏的HTML文件和引擎，直到实现游戏需要的所有功能为止。

期末考试

你有没有觉得我一下子给了你超多的信息呢？那就对了，我想要你在学习技能的同时有一些可以用来鼓捣的东西。为了完成这个习题，我将给你最后一套需要你自己完成的练习。你会注意到，到目前为止你写的游戏并不是很好，这只是你的第一版代码而已，你现在的任务就是让游戏更加完善，实现下面的这些功能。

1.修正代码中所有我提到和没提到的bug，如果你发现了新bug，你可以告诉我。

2.改进所有的自动测试，以便可以测试更多的内容，直到你可以不用浏览器就能测到所有的内容为止。

3.让HTML页面看上去更美观一些。

4.研究一下网页登录系统，为这个程序创建一个登录界面，这样人们就可以登录这个游戏，并且可以保存游戏高分。

5.完成游戏地图，尽可能地把游戏做大，功能做全。

6.给用户一个“帮助系统”，让他们可以查询每个房间里可以执行哪些命令。

7.为游戏添加新功能，想到什么功能就添加什么功能。

8.创建多个地图，让用户可以选择他们想要玩的一张地图来进行游戏。你的bin/app.py应该可以运行提供给它的任意地图，这样你的引擎就可以支持多个不同的游戏。

9.最后，使用在习题48和习题49中学到的东西创建一个更好的输入处理器。你手头已经有了大部分必要的代码，只需要改进语法，让它和你的输入表单以及游戏引擎挂钩即可。

祝你好运！

常见问题回答

我在游戏中用了会话（**session**），不能用**nosetests**测试。

你需要阅读并了解带reloader的会话：

http://webpy.org/cookbook/session_with_reloader。

我看到了**ImportError**。

错误路径，错误Python版本，PYTHONPATH没设置对，漏了__init__.py文件，拼写错误，都检查一下吧。

接下来的路

现在还不能说你是一名程序员。这本书的目的相当于给你一个“编程黑带”认证。你已经了解了足够的编程基础知识，并且有能力阅读别的编程书籍了。读完这本书，你应该已经掌握了一些学习的方法，并且具备了该有的学习态度，这样你在阅读其他Python书籍时也许会更顺利，而且能学到更多东西。

建议你看看下面这些项目，并试着用它们实现一些东西。

The Django Tutorial (<https://docs.djangoproject.com/en/1.4/intro/tutorial01>)，试着用 Django Web Framework 创建一个Web应用程序。

SciPy (<http://www.scipy.org>)，如果你对科学、数学、工程感兴趣可以看看。如果你想结合SciPy或者别的代码写篇美观的论文，还可以看看Dexy (<http://dexy.it>)。

PyGame (<http://www.pygame.org/news.html>)，看看能不能写出一个带图形界面和声音的游戏出来。

Pandas (<http://pandas.pydata.org>)，用来做数据处理和分析。

Natural Language Tool Kit (<http://nltk.org>)，用来分析文本，以及实现垃圾邮件过滤和自动聊天机器人这样的软件。

Requests (<http://docs.python-requests.org/en/latest/index.html>)，学习一下用户端HTTP以及Web知识。

SimpleCV (<http://simplecv.org>)，让你的计算机看到真实世界里的东西。

ScraPy (<http://scrapy.org>)，遍历并攫取网站内容。

Panda3D (<https://www.panda3d.org>)，设计3D图形界面和游戏。

Kivy (<http://kivy.org>)，桌面和移动平台的用户界面开发。

SciKit-Learn (<http://scikit-learn.org/stable>)，实现机器学习应用。

Ren'Py (<http://renpy.org>)，实现交互式角色扮演游戏，与本书中的游戏类似，不过多了图形界面。

Learn C The Hard Way (<http://c.learnthecodethehardway.org>)，等你熟悉Python后试着用我写的其他书学习C和算法。慢慢来，C是一门不同的语言，很值得学习。

选择一个项目，通读它的文档和简易教程。在阅读过程中将文档中的代码自己写一遍，并让它们正常运行。我是通过这样的方法学习的，其实每个程序员都是这么学的。读完教程和文档以后，试着写点东西出来。写什么都行，哪怕是别人写过的也可以，只要做出来东西就可以了。

你一开始写的东西可能很差，不过这没有关系。我在初学一种新语言时也是很菜的。没有哪个初学者能写出完美的代码来，如果有人告诉你他有这本事，那他只是在厚着脸皮撒谎而已。

怎样学习任何一种编程语言

我将教你怎样学习任何一种你将来可能要学习的编程语言。本书的章节编排是基于我和很多程序员学习编程的经历组织的，以下是我通常遵循的流程。

- 1.找到关于这种语言的书或介绍性读物。
- 2.通读这本书，把里边的代码写下来并运行起来。
- 3.一边读书一边写代码，同时做好笔记。
- 4.使用这种语言实现一些你用另一种熟悉的语言做过的程序组件。
- 5.阅读别人用这种语言写的代码，试着仿照他们的方式写代码。

在这本书里，我强制要求你慢慢地一点一点地完成了这个过程。别的书不是用这种方法写的，那就需要你把我教你的方法套用在这些书本上。最好的办法是先快速过一下书本内容，将里边主要的代码片段列出来，将这份列表变成一系列基于习题的章节，然后按照次序一一完成。

以上流程对于学习新技术也适用，只要有本相关的书，就能把它转换成这种练习格式。对于没有书的学习内容来说，你可以使用网上的教程或者源代码作为你的入门资料。

每学一种新的编程语言，你就会成长为一个更好的程序员。你学的越多，它们就会变得越容易学习。当你学到第三种或者第四种编程语言的时候，你就应该能够在一周内学会一门类似的语言，不过对于一些特别的语言来说你可能还是要花较长的时间。你现在学了Python，接下来学习Ruby和JavaScript就应该比较快了。这是因为很多语言有着共同的理念，你只要学了其中一种，就能用在别的语言上。

关于学习新语言的最后一件要记住的事情就是：别当一个蠢游客。

蠢游客就是那种去了一个国家旅游，然后回来抱怨那儿的饭不好吃的人。“为什么这个白痴国家连汉堡都买不到？”当你学习一种新语言时，不要假设它的工作方式太蠢——它只是不同而已——只有接受它你才能学会它。

不过，在学完一种语言后，不要成为这种语言工作方式的奴隶。有时你能看到有人使用一种语言做一些很白痴的事情，没有别的理由，只不过是“我以前一直就是这样做的”。如果你喜欢一种风格，而你又知道大家的做法和你不同，如果你看到后者能带来好处，那就毫不犹豫地打破自己的习惯吧。

我个人是很喜欢学习新编程语言的。我把自己当做一个“程序员人类学家”，我认为一种编程语言反映了一群使用者的一些独到见解。我学习的是他们用计算机互相交流时使用的语言，这对我来说非常有趣。不过话说回来，我这个人还是有点儿古怪的，所以对于新语言，你只要想学就学就行了。

好好享受吧！真的很有趣。

老程序员的建议

你已经完成了这本书并且打算继续编程。也许这会成为你的职业，也许你只是作为业余爱好，玩玩而已。无论如何，你都需要一些建议以保证你在正确的道路上继续前行，并且让这项新的爱好最大程度为你带来享受。

我做编程已经太长时间，长到对我来说编程已经是非常乏味的事情了。写这本书的时候，我已经懂得大约20种编程语言，而且可以在大约一天或者一个星期内学会一种编程语言（取决于这种语言有多古怪）。现在对我来说，编程这件事情已经很无聊，已经谈不上什么兴趣了。当然这不是说编程本身是一件无聊的事情，也不是说你以后也一定会这样觉得，这只是我个人在当前的感觉而已。

这么久的旅程下来，我的体会是：编程语言这东西并不重要，重要的是你用这些语言做的事情。事实上，我一直很清楚这一点，不过以前我会周期性地被各种编程语言分神而忘记了这一点。现在我是永远不会忘记这一点了，你也不应该忘记这一点。

你学的和用的编程语言并不重要。不要被围绕某一种语言的“宗教”把你扯进去，这只会让你忘掉语言的真正目的——作为你的工具来实现有趣的事情。

编程作为一项智力活动，是唯一一种能让你创建交互式艺术的艺术形式。你可以创建项目让别人使用，而且可以间接地和使用者沟通。没有其他的艺术形式能做到如此程度的交互性。电影引领观众走向一个方向，绘画是不会动的，而代码却是双向互动的。

编程作为一种职业只是一般有趣而已。编程可能是一份好工作，但如果你想赚更多的钱而且过得更快乐，其实开一间快餐分店就可以了。你最好的选择是将自己的编程技术作为自己的其他职业的秘密武器。

技术公司里边会编程的人多到一毛钱一打，根本得不到什么尊敬。而在生物学、医药学、政府部门、社会学、物理学、数学等行业领域从事编程的人就能得到足够的尊敬，而且你可以使用这项技能在这些领域做出令人惊异的成就。

当然，所有的这些建议都是无关紧要的。如果你跟着这本书学写软件而且觉得很喜欢这件事情的话，那你完全可以将其当作一种职业去追求。你应该继续深入拓展这个近五十年来极少有人探索过的奇异而美妙的智力工作领域。若能从中得到乐趣当然就更好了。

最后我要说的是，学习创造软件的过程会改变你，而让你与众不同。不是说更好了或更坏了，只是不同了。你也许会发现，因为你会写软件人们对你的态度有些怪异，也许会用“怪人”这样的词来形容你。也许你会发现，因为你会戳穿他们的逻辑漏洞而他们开始讨厌与你争辩。甚至你可能会发现，有人因为你懂得计算机怎么工作而觉得你是个讨厌的怪人。

对于这些我只有一个建议：让他们去死吧。这个世界需要更多的怪人，他们知道某样东西是怎么工作的而且喜欢找到答案。当有人那样对你时，只要记住这是你的旅程，不是他们的。“与众不同”不是谁的错，告诉你“与众不同是一种错”的人只是嫉妒你掌握了他们做梦都想不到的技能而已。

你会编程。他们不会。太酷了。

附录 命令行快速入门

这个附录是一个超快的命令行入门，你可以在一两天内读完这部分内容，这里不会教你命令行的高级应用。

简介：废话少说，命令行来也

这个附录会教你如何使用命令行来让你的计算机完成一些任务。作为一个快速 `rumen`，它的详细程度和我写的别的教程自然无法相比。它只是为了让你拥有基本足够的能力，从而可以开始像真正的程序员一样使用计算机。读完这个附录以后，你将学会命令行使用者每天使用的大部分基本命令，而且你将能基本理解目录以及一些别的概念。

我给你的唯一一个建议是：废话少说，动手把这些都键入进去。

话是刻薄了点儿，但这就是你要做的。如果你对命令行有一种非理性的恐惧，克服恐惧的唯一办法就是废话少说，和它斗到底。

你不会把自己的计算机弄坏。你不会被抓起来关到微软总部的底下秘密监牢里。你的朋友不会笑话你是个计算机呆子。所有那些害怕命令行的理由，你都忽略掉吧。

为什么呢？因为如果要学习编程，你就必须学习命令行。编程语言是控制计算机的进阶方式，命令行则算是编程语言的小弟。一旦越过这道坎，你就可以继续学习编程，并且你会感觉到，你买的这一台沉甸甸的机器总算真正属于你了。

如何使用这个附录

最好的办法是照下面的方法来做。

准备一个小笔记本和一支笔。

按照书中的方法完成每一道习题。

遇到不懂的或者无法理解的东西，就把它记录在笔记本上，并在问题的下面留下一部分空间，以供日后作答。

完成一个习题后，过一遍你在笔记本中记录的问题。先试着通过网上搜索的方法解决你的问题，或者问一下懂的朋友也可以。你也可以写邮件给我，我也可以提供帮助。

在做每一个习题的过程中都重复上述步骤。到最后，你对命令行的了解将大大超过自己的预期。

你需要发挥记忆力

在一切开始之前先提醒你一下，我会马上让你开始记一些东西。这是让你上手的最快方法。对于一些人来说，记东西是很痛苦的一件事情，这就需要你披荆斩棘，坚持到底。记是一个很重要的学习技能，所以你要克服自己的恐惧。

现在告诉你怎样记东西。

告诉自己你一定会去做。不要试图去找技巧或者小窍门什么的，安心去做这件事就好了。

把你要记的东西写在索引卡上。要学的东西写在一面，答案写在另一面。

每天花15~30分钟，专心学习你的索引卡，试着把每一张都记住。把你没记住的卡片单独放一摞，没事干的时候就专门学习一下。最后把所有卡放一整摞，测验一下自己提高了多少。

晚上睡觉前花5分钟学习一下自己答错的卡片。

还有一些别的记忆技巧，比如你可以把要记的东西写在一张纸上，然后贴在浴室的墙上。当你洗澡的时候，你可以试着不看答案回想你的学习内容，如果在哪里卡住了，你可以瞟一眼答案刷新一下记忆。

如果每天都照着上面的步骤做，你应该在一周或者最多一个月的时间内记住这些东西。一旦记忆的工作完成后，其他的一切就更容易了，这也是记的目的。记不是为了让你理解抽象概念，而是让你把细节印在

大脑里，下次遇到时就不用去想它了。一旦你记住了这些基础知识，它们就不再会是你学习更高级的抽象概念的阻碍了。

习题1 准备工作

这本书将带领你做以下三件事情。

在你的shell（或命令行，或者Terminal，或者PowerShell）写一些东西。

弄懂你刚写的东西。

自己再多写一些东西。

第一个习题中，你的目的是打开自己的终端并确认它能正常工作，以便继续学习下去。

任务

准备好你的Terminal、shell和PowerShell，设置好，以便快速访问。

Mac OSX

对于Mac OSX你要做的具体如下。

按住command键，同时敲击空格键。

右上方会跳出蓝色的“搜索栏”。

键入“terminal”。

点击长得像一个黑盒子的终端应用程序。

这样终端就打开了。

你可以按着Ctrl点击Dock，在打开的菜单中选择“Options → Keep”，这样终端就会一直保留在Dock里了。

这样你就打开了终端，而且Dock里也有了快速访问链接。

Linux

如果你已经在使用 Linux，那我就可以假设你已经知道如何找到终

端了。在你的窗口管理器（window manager）里搜寻名字像“Shell”或者“Terminal”的东西就可以了。

Windows

Windows 下我们将使用 PowerShell。人们以前用的是一个叫cmd.exe 的程序，不过和PowerShell比起来它的可用性差很多。如果你用的是Windows 7以上的系统，就照下面的做。

点击Windows的“开始”菜单。

在搜索框中键入“powershell”。

敲回车键。

如果你装的不是Windows 7，那应该认真考虑一下升级事宜。如果你实在不想升级，那就试着从微软公司的下载中心安装一下。这得靠你自己了，因为我没有装Windows XP，写不出安装流程来，不过希望Windows XP下的PowerShell的使用体验也是一样的。

[知识点](#)

你学会了如何打开你的终端，这是继续本书所必需的。

注意 如果你有一个挺聪明而且懂Linux的朋友，再如果他们介绍bash之外的shell给你，那你应该忽略他们的建议。我教你的是bash，就是这样。他们会说zsh会让你的IQ多长30个点，并且让你在股票市场上赚得腰缠万贯，别理他们就是了。你的目的只是学会足够的技能，在你这个技能等级上，使用哪个shell其实不会影响什么。还有就是，躲开IRC以及那些“黑客”常去的地方。他们会教你一些破坏你电脑的命令并以此为乐。例如，这条经典的rm -rf /，千万别输这条命令！离黑客远点，如果你需要帮助，就找你能信任的人，别去网上找。

[更多任务](#)

这个习题有一个很大的“更多任务”部分。其他的练习没这么多的额外任务要做，不过我要让你通过记忆的方式向自己灌输足够的知识。跟着我走就行了，这会让你后面的学习变得非常顺畅。

Linux/Mac OSX

用索引卡片写下所有的命令，一张卡片写一条，正面写下命令，背面写下解释。每次看书就学习一次，每次学个15分钟左右。

`pwd`（打印工作目录）

`hostname`（电脑在网络中的名称）

`mkdir`（创建路径）

`cd`（更改路径）

`ls`（列出路径下的内容）

`rmdir`（删除路径）

`pushd`（推入路径）

`popd`（弹出路径）

`cp`（复制文件或路径）

`mv`（移动文件或路径）

`less`（逐页浏览文件）

`cat`（打印输出整个文件）

`xargs`（执行参数）

`find`（寻找文件）

`grep`（在文件中查找内容）

`man`（阅读手册）

`apropos`（寻找恰当的手册页面）

`env`（查看你的环境）

`echo`（打印一些参数）

`export`（导出/设定一个新的环境变量）

`exit`（离开shell）

`sudo`（成为超级用户或root，危险命令！）

Windows

如果用的是Windows，下面是你要学习的命令。

`pwd`（打印工作目录）

`hostname`（电脑在网络中的名称）

`mkdir`（创建路径）

`cd`（更改路径）

`ls`（列出路径下的内容）

`rmdir`（删除路径）

`pushd`（推送路径）

`popd`（弹出路径）

`cp`（复制文件或路径）

`robocopy`（更可靠的复制命令）

`mv`（移动文件或路径）

`more`（逐页显示整个文件）

`type`（打印输出整个文件）

`forfiles`（在一大堆文件上面运行一条命令）

`dir -r`（寻找文件）

`select-string`（在文件中查找内容）

`help`（阅读手册）

`helpctr`（寻找恰当的手册页面）

`echo`（打印一些参数）

`set`（导出/设定一个新的环境变量）

`exit`（退出shell）

`runas`（成为超级用户或root，危险命令！）

不停地练习，直到你能做到：看到一条命令，然后能立即说出它的功能为止；反过来也能说出实现每个功能所需的命令。通过这样的方式

你可以为自己建立语汇，不过如果你觉得烦，也别强迫自己在上面花太多时间。

习题2 路径、文件夹和目录（pwd）

这个习题将让你学会使用pwd命令来打印你的工作目录。

任务

接下来我要教你如何阅读我展示给你的这些终端会话。你不需要将所有的东西都键入终端，只要键入其中的一部分内容而已。

你不需要键入\$（Unix）或者>（Windows），它们只是命令行终端会话的一个标记。

写完\$或者>后面的内容后需要敲击回车键。所以，如果你看到了\$ pwd，就需要键入pwd再敲击一次回车键。

你可以看到输出的内容前面也有一个\$或者>提示。这些是输出内容，你的输出内容和我的应该是一样的。

让我们先试一个命令，看看你有没有弄明白。

习题2 Linux/Mac OSX会话

```
$ pwd
/Users/zedshaw
$
```

习题2 Windows会话

```
PS C:\Users\zed> pwd
Path
----
C:\Users\zed
PS C:\Users\zed>
```

注意 为了节约空间同时让你集中精力到重要的命令细节上面，本附录将把命令行一开始的部分（如上面的PS C:\Users\zed）省略掉，只留下一个小小的>部分。这意味着，你的命令行和这里看到的会有一点不同，不过这是正常的，你无须操心。记住，从现在开始，我只通过>来告诉你这是一个命令行提示。对于Unix命令行提示也一样，不过Unix有点不一样，人们习惯使用\$来表示命令提示符。

知识点

你的命令行和我的看上去不一样，你的可能在\$前面打印了你的用户名以及计算机名。Windows下看上去也会不一样，不过关键的基本格式都是下面这样的。

有一个命令提示符。

键入一条命令，如这里的pwd。

得到一些输出。

重复上述步骤。

你正好还学会了pwd的功能，它的意思是“打印工作目录”。目录是什么东西？就是文件夹而已。文件夹和目录是一样的东西，这两个词可以互相替换。如果你打开文件浏览器，通过图形界面寻找文件，那你就是在访问文件夹。这些文件夹和我们后面要用到的目录完全是一回事儿。

更多任务

键入20次pwd，每次键入都念一遍“print working directory”。

记下命令行打印的路径，用你的文件浏览器找到这个位置。

我不是开玩笑。写20遍，并且朗读出来。别抱怨了，照我说的做。

习题3 如果你迷失了

在你学习的过程中，你也许会迷失在命令行里。你也许不知道自己的当前位置，或者找不到某个文件，然后就不知道接下来怎么做。为解决这个问题，我将教你一个不迷失的命令。

迷失的原因通常是你键入了一些命令，然后就不知道自己最后跑到哪个目录下了。这时你应该做的是键入`pwd`打印出你的当前路径，然后你就知道自己的位置了。

接下来你需要一个回到安全路径（也就是你的`home` 路径）的方法。很简单，键入 `cd ~`就可以了。

也就是说，如果你下次迷失了，只要键入：

```
pwd
```

```
cd~
```

第一个命令`pwd`告诉你你当前的位置，第二个命令`cd ~`将你带回`home`路径。

任务

使用`pwd`和`cd ~`弄清楚自己的当前路径，然后回到`home`路径。确保自己总在正确的目录里。

知识点

你学会了在迷路后怎样回家。

习题4 创建目录（mkdir）

这个习题将让你学会怎样使用mkdir命令来创建新目录（文件夹）。

任务

习题4 Linux/Mac OSX会话

```
$ pwd
$ cd ~
$ mkdir temp
$ mkdir temp/stuff
$ mkdir temp/stuff/things
$ mkdir -p temp/stuff/things/frank/joe/alex/john
$
```

习题4 Windows会话

```
> pwd
> cd ~
> mkdir temp
Directory: C:\Users\zed
Mode                LastWriteTime         Length Name
----                -
d-----           12/17/2011   9:02 AM             temp
> mkdir temp/stuff
Directory: C:\Users\zed\temp
```

```

Mode                LastWriteTime         Length Name
----                -
d----          12/17/2011   9:02 AM             stuff
> mkdir temp/stuff/things
    Directory: C:\Users\zed\temp\stuff
Mode                LastWriteTime         Length Name
----                -
d----          12/17/2011   9:03 AM             things
> mkdir temp/stuff/things/frank/joe/alex/john
    Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex
Mode                LastWriteTime         Length Name
----                -
d----          12/17/2011   9:03 AM             john
>

```

知识点

现在我们键入了好几条命令。这些命令是使用mkdir的不同方法。mkdir的功能是什么呢？它是用来创建目录（make directory）的。不该问这个问题吧？你应该已经通过索引卡记住这些了才对。如果不知道这一条，就说明你需要继续在索引卡上下功夫。

创建目录是什么意思？目录又可以叫作“文件夹”，它们是一回事儿。你上面所做的是在逐层深入地创建目录，目录有时又叫做路径，这里相当于是说“先到 temp，再到 stuff，然后到things，这就是我要到的地方。”这是对计算机发出的一系列指向，告诉计算机你想要把某个东西放到计算机硬盘的某个文件夹（路径）里。

注意 本书中我使用斜杠（/）来表示路径，因为所有的计算机都是

这么做的。不过，Windows用户应该知道，反斜杠（\）也可以实现同样的功能，别的Windows用户可能认为这才是正常的用法。

[更多任务](#)

你可能觉得路径的概念还是有些绕。别担心，我们会做大量的练习让你深入理解。

在temp下面再创建20个别的目录，深度可以各不相同，然后用文件浏览器检查你创建的目录。

创建一个名称包含空格的目录，方法是为名称添加一个引号：
`mkdir "I Have Fun"`。

如果目录已经存在，要创建它时将会得到一条错误信息。使用cd转到一个别的目录下面试试创建temp目录，如果你用Windows的话，桌面（desktop）是个不错的选择。

习题5 更改目录 (cd)

这个习题将教会你如何使用cd命令来更改目录。

任务

我将再教一遍终端会话的方法。

\$ (Unix) 和> (Windows) 是不需要写出来的。

你写完\$或>后面的内容后需要敲回车键。如果你看到\$ cd temp, 你需要键入的就是cd temp然后敲回车键。

敲回车后你会看到输出，输出的前面也会有一个\$或者>提示符。

每次开始练习前都先进入home路径。键入pwd然后用cd ~回到你的起始位置。

习题5 Linux/Mac OSX会话

```
$ cd temp
```

```
$ pwd
```

```
~/temp
```

```
$ cd stuff
```

```
$ pwd
```

```
~/temp/stuff
```

```
$ cd things
```

```
$ pwd
```

```
~/temp/stuff/things
```

```
$ cd frank/
```

```
$ pwd
```

```
~/temp/stuff/things/frank
$ cd joe/
$ pwd
~/temp/stuff/things/frank/joe
$ cd alex/
$ pwd
~/temp/stuff/things/frank/joe/alex
$ cd john/
$ pwd
~/temp/stuff/things/frank/joe/alex/john
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things/frank/joe
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things
$ cd ../../..
$ pwd
~/
$ cd temp/stuff/things/frank/joe/alex/john
$ pwd
~/temp/stuff/things/frank/joe/alex/john
$ cd ../../../../../../../
$ pwd
~/
```

\$

习题5 Windows会话

> cd temp

> pwd

Path

C:\Users\zed\temp

> cd stuff

> pwd

Path

C:\Users\zed\temp\stuff

> cd things

> pwd

Path

C:\Users\zed\temp\stuff\things

> cd frank

> pwd

Path

C:\Users\zed\temp\stuff\things\frank

> cd joe

> pwd

Path

C:\Users\zed\temp\stuff\things\frank\joe

```
> cd alex
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank\joe\alex
```

```
> cd john
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank\joe\alex\john
```

```
> cd ..
```

```
> cd ..
```

```
> cd ..
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank
```

```
> cd ../../
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff
```

```
> cd ..
```

```
> cd ..
```

```
> cd temp/stuff/things/frank/joe/alex/john
```

```
> cd ../../../../../../../
```

```
> pwd
```

Path

C:\Users\zed

>

知识点

你在上一个习题中创建了不少的目录，现在你所做的就是通过 `cd` 命令在它们之间往来。在上面的终端会话中，我通过使用 `pwd` 来检查自己的当前路径，所以，请记住别把 `pwd` 的输出也当作要键入的东西。例如，第三行有一条 `~/temp`，但这只是 `pwd` 的输出而已，别把它也键入了。

你应该还看到我可以使用 `..` 移动到目录的上一层。

更多任务

在图形界面计算机上学习使用命令行界面（`command line interface`, CLI）很重要的一部分，就是弄明白命令行和图形界面是如何互相配合工作的。我开始使用计算机的时候，GUI还不存在，所有的事情都要通过DOS命令窗口（CLI）来完成。后来计算机越变越强大，人人都能用到图形界面了。对我来说，命令行的目录和图形界面的文件夹都很容易理解。

然而现在大部分的人都不理解命令行界面以及路径和目录这些概念。其实这些东西也很难学会，只有不停地学习和使用CLI，直到有一天豁然开朗，所有在GUI下做的事情都和CLI下要做的对应起来了。

早日理解的办法是花一些时间来通过图形界面的文件浏览器来寻找目录，然后通过命令行去访问它们，这就是你接下来要做的练习。

用一条命令 `cd` 到 `joe` 目录。

用一条命令回到temp目录，不过不要退得太远了。

找出如何用一条命令cd到你的“home目录”。

cd 到你的 Documents 目录，然后通过你的图形文件浏览器找到这个目录（Finde、Windows浏览器等）。

cd到你的Downloads目录，然后通过文件浏览器找到这个位置。

用文件浏览器找到另外一个位置，然后cd到这个位置。

还记得你可以为包含空格的目录加一个引号吧？对于任何命令，你都可以这么做。假如你创建了一个叫 I Have Fun的文件夹，你就可以使用 cd "I Have Fun"这条命令。

习题6 列出目录下的内容 (ls)

本节将教你怎样用ls命令列出目录下的内容。

任务

开始之前，确认你已经到了temp的上一级目录。如果不确定现在在哪个目录里，就用pwd找出来。

习题6 Linux/Mac OSX会话

```
$ cd temp
$ ls
stuff
$ cd stuff
$ ls
things
$ cd things
$ ls
frank
$ cd frank
$ ls
joe
$ cd joe
$ ls
alex
$ cd alex
```

```
$ ls
$ cd john
$ ls
$ cd ..
$ ls
john
$ cd ../../../
$ ls
frank
$ cd ../../
$ ls
stuff
$
```

习题6 Windows会话

```
> cd temp
> ls

Directory: C:\Users\zed\temp
Mode                LastWriteTime         Length Name
----                -
d----          12/17/2011   9:03 AM             stuff
> cd stuff
> ls

Directory: C:\Users\zed\temp\stuff
Mode                LastWriteTime         Length Name
----                -
d----          12/17/2011   9:03 AM             things
> cd things
```


> ls

Directory: C:\Users\zed\temp\stuff\things

Mode		LastWriteTime	Length	Name
d----	12/17/2011	9:03 AM		frank

> cd frank

> ls

Directory: C:\Users\zed\temp\stuff\things\frank

Mode		LastWriteTime	Length	Name
d----	12/17/2011	9:03 AM		joe

> cd joe

> ls

Directory: C:\Users\zed\temp\stuff\things\frank\joe

Mode		LastWriteTime	Length	Name
d----	12/17/2011	9:03 AM		alex

> cd alex

> ls

Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode		LastWriteTime	Length	Name
d----	12/17/2011	9:03 AM		john

> cd john

> ls

> cd ..

> ls

```

    Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex
Mode                LastWriteTime         Length Name
----                -
d----             12/17/2011   9:03 AM             john
> cd ..
> ls

    Directory: C:\Users\zed\temp\stuff\things\frank\joe
Mode                LastWriteTime         Length Name
----                -
d----             12/17/2011   9:03 AM             alex
> cd ../../..
> ls

    Directory: C:\Users\zed\temp\stuff
Mode                LastWriteTime         Length Name
----                -
d----             12/17/2011   9:03 AM             things
> cd ..
> ls

    Directory: C:\Users\zed\temp
Mode                LastWriteTime         Length Name
----                -
d----             12/17/2011   9:03 AM             stuff
>

```

知识点

ls命令列出了你当前所在的目录下的内容。我使用了cd变更目录，

然后列出里边的内容，这样我就知道接下来该到哪个目录下面去了。

`ls`命令有很多选项，不过后面你会通过学习来自己发现这些。

[更多任务](#)

键入每一条命令！要学习这些命令，你必须键入这些命令。光读是不够的。这一点我以后就不跟你啰嗦了。

如果你用Unix，那就在目录中试一下`ls -lR`命令。

Windows下一样的功能可以通过`dir -R`完成。

使用`cd`进到别的目录下，然后通过`ls`看看里边有什么内容。

在笔记本上记下你的问题。我知道你会有些问题，因为对于这条命令我并没讲全。

记住，如果你在路径中迷失了，就使用`ls`和`pwd`找出你的当前路径，然后通过`cd`到达你的目的路径即可。

习题7 删除路径（**rmdir**）

在这个习题中你将学会怎样删除一个空目录。

任务

习题7 Linux/Mac OSX会话

```
$ cd temp
$ ls
stuff
$ cd stuff/things/frank/joe/alex/john/
$ cd ..
$ rmdir john
$ cd ..
$ rmdir alex
$ cd ..
$ ls
joe
$ rmdir joe
$ cd ..
$ ls
frank
$ rmdir frank
$ cd ..
$ ls
```

```

things
$ rmdir things
$ cd ..
$ ls
stuff
$ rmdir stuff
$ pwd
~/temp
$

```

警告 如果在Mac OSX上做rmdir并且遇到你确定是空目录但它拒绝删除该目录的情况，实际上在这个目录中有一个名为.DS_Store 的文件。这种情况下，键入 `rm-rf <dir>`即可（<dir>用实际的目录名代替）。

习题7 Windows会话

```

> cd temp
> ls

```

Directory: C:\Users\zed\temp			
Mode		LastWriteTime	Length Name
----		-----	-----
d----	12/17/2011	9:03 AM	stuff

```

> cd stuff/things/frank/joe/alex/john/
> cd ..
> rmdir john
> cd ..
> rmdir alex
> cd ..
> rmdir joe
> cd ..

```

```

> rmdir frank
> cd ..
> ls
    Directory: C:\Users\zed\temp\stuff
Mode                LastWriteTime         Length Name
----                -
d-----          12/17/2011   9:14 AM             things
> rmdir things
> cd ..
> ls
    Directory: C:\Users\zed\temp
Mode                LastWriteTime         Length Name
----                -
d-----          12/17/2011   9:14 AM             stuff
> rmdir stuff
> pwd
Path
----
C:\Users\zed\temp
> cd ..
>

```

知识点

我现在将命令混到了一起，所以你要集中精力确认键入完全相同。你的每一个错误都是不够集中精力导致的。如果你发现自己犯了很多错误，那就休息一会儿，或者今天就别接着学习了，等明天再鼓足精神继

续。

在本例中你学会了如何删除（**remove**）一个目录。这很容易，你只要到你要移除的目录的上一层，然后键入`rmdir <dir>`，将`<dir>`替换成你要删除的目录的名称即可。

更多任务

再创建20个目录，然后把它们都删除掉。

创建一个逐层嵌套的目录，一共嵌套10层，然后进到目录中将这些目录逐层删掉，就更我上面做的一样。

如果你要移除的目录中包含一些内容，就会遇到一个错误信息。后面的练习中我会告诉你如何移除这样的目录。

习题8 在多个目录中切换（pushd, popd）

这个习题中你将学会怎样使用 pushd 保存当前路径并转到一个新路径下，以及怎样通过popd回到先前保存的路径下去。

任务

习题8 Linux/Mac OSX会话

```
$ cd temp
$ mkdir -p i/like/icecream
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ popd
~/temp
$ pwd
~/temp
$ pushd i/like
~/temp/i/like ~/temp
$ pwd
~/temp/i/like
$ pushd icecream
~/temp/i/like/icecream ~/temp/i/like ~/temp
$ pwd
~/temp/i/like/icecream
$ popd
```



```

~/temp/i/like ~/temp
$ pwd
~/temp/i/like
$ popd
~/temp
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ pushd
~/temp ~/temp/i/like/icecream
$ pwd
~/temp
$ pushd
~/temp/i/like/icecream ~/temp
$ pwd
~/temp/i/like/icecream
$

```

习题8 Windows会话

```

> cd temp
> mkdir -p i/like/icecream
    Directory: C:\Users\zed\temp\i\like
Mode                LastWriteTime         Length Name
----                -
d-----          12/20/2011  11:05 AM             icecream
> pushd i/like/icecream
> popd
> pwd
Path

```

```
----  
C:\Users\zed\temp  
> pushd i/like  
> pwd  
Path  
----  
C:\Users\zed\temp\i\like  
> pushd icecream  
> pwd  
Path  
----  
C:\Users\zed\temp\i\like\icecream  
> popd  
> pwd  
Path  
----  
C:\Users\zed\temp\i\like  
> popd  
>
```

知识点

如果你用到这些命令，那你就非常接近编程领域了。这些命令非常好用，所以我非教你不可。这些命令可以让你临时跑到某个不同的目录中，然后再回到之前的目录，并且方便地在两者之间切换。

`pushd`命令会将你目前所在的目录“推送”（`push`）到一个列表中以供后续使用，然后让你转入到另一个目录中。它的意思大致是：“记住

我现在的位置，然后到这个地方去。”

`popd`命令会将你上次推送过的目录从列表中“弹出”（`pop`），然后让你回到这个被“弹出”的目录。

最后，在 `Unix` 中有点不同，如果运行时不添加任何参数，那么它就会让你在当前目录和你上一次推送过的目录之间切换，这个方法可以让你很方便地在两个目录之间切换。不过 `PowerShell` 中这样做是不灵的。

[更多任务](#)

使用这些命令在你的计算机目录之间多切换几次。

删掉 `i/like/icecream` 这一系列目录，然后自己创建一些目录，在它们之间切换。

向自己解释 `pushd` 和 `popd` 的输出的意义。有没有发现它的工作模式有点像一个栈？

前面已经教过了，但要记住，`mkdir -p` 会创建一个完整的多层目录，即使中间目录不存在也能成功。这也是我创建这个习题一开始所做的事情。

习题9 创建空文件（touch, New-Item）

这个习题你将学会怎样使用touch（Windows中是New-Item）命令创建一个空文件。

任务

习题9 Linux/Mac OSX会话

```
$ cd temp
$ touch iamcool.txt
$ ls
iamcool.txt
$
```

习题9 Windows会话

```
> cd temp
> New-Item iamcool.txt -type file
> ls

Directory: C:\Users\zed\temp

Mode                LastWriteTime         Length Name
----                -
-a---             12/17/2011   9:03 AM             iamcool.txt
>
```

知识点

你学会了如何创建空文件。在Unix中你要用touch，它还有一个功

能就是修改文件的时间。我除了用它创建空文件以外，很少用它做别的事情。Windows中没有这个命令，所以你要学习使用New-Item命令，它实现的功能是一样的，只不过它还可以创建新目录。

[更多任务](#)

Unix: 创建一个目录，转到这个目录下，然后在其中创建一个文件，然后回到上一层目录，对你创建的目录运行rmdir，你应该会看到一个错误，试着弄懂为什么你会遇到这个错误。

Windows: 做同样的事情，不过你不会看到错误，你会看到一个提示，问你是否真的要删除这个目录。

习题10 复制文件（cp）

这个习题你将学会使用cp命令将文件从一个地方复制（copy）到另一个地方。

任务

习题10 Linux/Mac OSX会话

```
$ cd temp
$ cp iamcool.txt neat.txt
$ ls
iamcool.txt neat.txt
$ cp neat.txt awesome.txt
$ ls
awesome.txt iamcool.txt  neat.txt
$ cp awesome.txt thefourthfile.txt
$ ls
awesome.txt  iamcool.txt    neat.txt    thefourthfile.txt
$ mkdir something
$ cp awesome.txt something/
$ ls
awesome.txt  iamcool.txt    neat.txt    something
thefourthfile.txt
$ ls something/
awesome.txt
```

```
$ cp -r something newplace
```

```
$ ls newplace/
```

```
awesome.txt
```

```
$
```

习题10 Windows会话

```
> cd temp
```

```
> cp iamcool.txt neat.txt
```

```
> ls
```

```
Directory: C:\Users\zed\temp
```

Mode		LastWriteTime	Length	Name
----		-----	-----	----
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt

```
> cp neat.txt awesome.txt
```

```
> ls
```

```
Directory: C:\Users\zed\temp
```

Mode		LastWriteTime	Length	Name
----		-----	-----	----
-a---	12/22/2011	4:49 PM	0	awesome.txt
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt

```
> cp awesome.txt thefourthfile.txt
```

```
> ls
```

```
Directory: C:\Users\zed\temp
```

Mode		LastWriteTime	Length	Name
----		-----	-----	----
-a---	12/22/2011	4:49 PM	0	awesome.txt

```
-a---      12/22/2011   4:49 PM          0 iamcool.txt
-a---      12/22/2011   4:49 PM          0 neat.txt
-a---      12/22/2011   4:49 PM          0 thefourthfile.txt
```

> mkdir something

Directory: C:\Users\zed\temp

```
Mode                LastWriteTime         Length Name
----                -
d----      12/22/2011   4:52 PM              something
```

> cp awesome.txt something/

> ls

Directory: C:\Users\zed\temp

```
Mode                LastWriteTime         Length Name
----                -
d----      12/22/2011   4:52 PM              something
-a---      12/22/2011   4:49 PM          0 awesome.txt
-a---      12/22/2011   4:49 PM          0 iamcool.txt
-a---      12/22/2011   4:49 PM          0 neat.txt
-a---      12/22/2011   4:49 PM          0 thefourthfile.txt
```

> ls something

Directory: C:\Users\zed\temp\something

```
Mode                LastWriteTime         Length Name
----                -
-a---      12/22/2011   4:49 PM          0 awesome.txt
```

> cp -recurse something newplace

> ls newplace

Directory: C:\Users\zed\temp\newplace

```
Mode                LastWriteTime         Length Name
```



```
-----  
-a---      12/22/2011   4:49 PM           0 awesome.txt  
>
```

知识点

现在你学会了复制文件。很简单，就是把一个文件复制成一个新文件而已。在这个习题中我还创建了一个新目录，然后将文件复制到其中去。

我现在要告诉你一个关于程序员和系统管理员的秘密：他们都很懒。我是懒人，我的朋友们也是懒人，这也是我们用计算机的原因。我们让计算机为我们做各种无聊的事情。你学到现在，所做的事情就是重复键入各种无趣的命令，并通过这个过程学会这些命令，但实际工作中不是这样子的。在实际工作中，如果你发现某个任务需要通过无趣的重复工作来完成，那么很可能已经有程序员找出让这个任务变得更简单的方法了，只不过你不知道而已。

另外要告诉你的就是：程序员其实没有你想象的那么聪明。如果你过度思考要键入的命令名，结果很可能是过而不及。相反，你应该去想这个命令的名字，然后直接试试这个名字或者类似这个名字的缩写。如果还是不灵，那就问问别人，或者上网搜索。不过遇到ROBOCOPY这么傻的命令名就真没什么好办法记住了。

更多任务

练习使用cp命令复制一些包含文件的目录。

将一个文件复制到你的home目录或者桌面。

从图形界面找到你复制过的文件，然后用文本编辑器打开它们。

有没有发现我有时会在目录的结尾放一个斜杠 (/)？这样做的目

的是保证键入的名称确实是一个目录，于是如果这个目录不存在，那么我就会看到一个错误信息。

习题11 移动文件 (mv)

这个习题你将学会怎样使用mv命令把文件从一个地方移动到另一个地方。

任务

习题11 Linux/Mac OSX会话

```
$ cd temp
$ mv awesome.txt uncool.txt
$ ls
newplace      uncool.txt
$ mv newplace oldplace
$ ls
oldplace      uncool.txt
$ mv oldplace newplace
$ ls
newplace      uncool.txt
$
```

习题11 Windows会话

```
> cd temp
> mv awesome.txt uncool.txt
> ls

Directory: C:\Users\zed\temp

Mode                LastWriteTime         Length Name
-----
```

```

-----
d----      12/22/2011   4:52 PM           newplace
d----      12/22/2011   4:52 PM           something
-a---      12/22/2011   4:49 PM       0 iamcool.txt
-a---      12/22/2011   4:49 PM       0 neat.txt
-a---      12/22/2011   4:49 PM       0 thefourthfile.txt
-a---      12/22/2011   4:49 PM       0 uncool.txt

```

> mv newplace oldplace

> ls

Directory: C:\Users\zed\temp

```

Mode                LastWriteTime         Length Name
-----
d----      12/22/2011   4:52 PM           oldplace
d----      12/22/2011   4:52 PM           something
-a---      12/22/2011   4:49 PM       0 iamcool.txt
-a---      12/22/2011   4:49 PM       0 neat.txt
-a---      12/22/2011   4:49 PM       0 thefourthfile.txt
-a---      12/22/2011   4:49 PM       0 uncool.txt

```

> mv oldplace newplace

> ls newplace

Directory: C:\Users\zed\temp\newplace

```

Mode                LastWriteTime         Length Name
-----
-a---      12/22/2011   4:49 PM       0 awesome.txt

```

> ls

Directory: C:\Users\zed\temp

```

Mode                LastWriteTime         Length Name

```

```
-----
d----      12/22/2011   4:52 PM           newplace
d----      12/22/2011   4:52 PM           something
-a---      12/22/2011   4:49 PM       0 iamcool.txt
-a---      12/22/2011   4:49 PM       0 neat.txt
-a---      12/22/2011   4:49 PM       0 thefourthfile.txt
-a---      12/22/2011   4:49 PM       0 uncool.txt
>
```

知识点

移动（move）文件，或者换种说法，重命名（rename）文件。很简单，给出旧文件名和新文件名即可。

更多任务

将一个文件从newplace目录移动到另一个目录，然后再将它移动回来。

习题12 查看文件内容（less, MORE）

要完成这个习题，你将用到已经学过的一些命令，另外还需要一个文本编辑器来创建纯文本（.txt）文件，下面是要做的准备工作。

打开文本编辑器，在新文本中键入一些东西。在 OSX 中你可以用 TextWrangler，在Windows下可以用Notepad++，在Linux中可以用 gedit，随便什么编辑器都可以。

保存该文件到桌面，将其命名为test.txt。

在shell中使用学过的命令将该文件复制到你的工作目录，也就是 temp 目录中去。做好准备工作以后，就可以完成任务了。

任务

习题12 Linux/Mac OSX会话

```
$ less test.txt  
[displays file here]  
$
```

就是这样子。要退出tess，只要键入q即可。这个q指的就是quit。

习题12 Windows会话

```
> more test.txt  
[displays file here]  
>
```

注意 上面的输出中我用[displays file here]来指代程序的输出。在后面的练习中，如果遇到复杂情况无法向你展示输出内容，我就会用这个来指代你的输出。你的屏幕不会显示这句话。

知识点

这是查看文件内容的一个方法。它有用的地方在于，如果文件内容有很多行，它会将其分页，这样就会每次显示一页。在“更多任务”中你会看到更多相关的练习。

更多任务

再次打开你的文本文件，重复复制粘贴若干次，让你的文本长度约等于50~100行。

将它再次复制到temp目录下，这样你就可以通过命令行查看了。

再做一遍这个习题，不过这次你要逐页浏览文档。在Unix下使用空格键和w键上下翻页，使用方向键也可以，不过在Windows下就只能用空格键向下逐页浏览了。

查看你创建的空文件的内容。

命令会覆盖已经存在的文件，复制文件时要留意这一点。

习题13 流文件内容显示 (cat)

你需要更多的准备工作，这个过程也会让你习惯这个工作流程：你
在一个程序中创建文件，然后通过命令行对其进行处理。使用习题12中
的文本编辑器创建一个叫test2.txt的文件，只是这一次要将其直接保存到
temp目录下。

任务

习题13 Linux/Mac OSX会话

```
$ less test2.txt
[displays file here]
$ cat test2.txt
I am a fun guy.
Don't you know why?
Because I make poems,
that make babies cry.
$ cat test.txt
Hi there this is cool.
$
```

习题13 Windows会话

```
> more test2.txt
[displays file here]
> cat test2.txt
I am a fun guy.
```



```
Don't you know why?  
Because I make poems,  
that make babies cry.
```

```
> cat test.txt
```

```
Hi there this is cool.
```

```
>
```

记住，我写的[displays file here]是表示我略掉了命令的输出，这样我就不用把东西详尽地展示给你了。

[知识点](#)

我的诗怎么样？拿个诺贝尔奖没问题吧？不管怎样，你已经学了第一个命令，而我只是让你检查你的文件已经在那里了。然后你使用`cat`将文件内容显示到屏幕上。这个命令会将整个文件一次输出到屏幕，不会分页也不会中间停顿。为了展示这一点，我让你对 `test2.txt` 执行这个命令，结果就是一次输出了文本中所有的行。

[更多任务](#)

再创建几个文本文件，然后用逐一打开。

Unix：试试`cat ex12.txt ex13.txt`，看看结果是怎样的。

Windows：试试`cat ex12.txt,ex13.txt`，看看结果是怎样的。

习题14 删除文件 (rm)

这个习题将让你学会怎样使用rm命令删除文件。

任务

习题14 Linux/Mac OSX会话

```
$ cd temp
$ ls
uncool.txt  iamcool.txt  neat.txt  something  thefourthfile.txt
$ rm uncool.txt
$ ls
iamcool.txt  neat.txt  something  thefourthfile.txt
$ rm iamcool.txt neat.txt thefourthfile.txt
$ ls
something
$ cp -r something newplace
$
$ rm something/awesome.txt
$ rmdir something
$ rm -rf newplace
$ ls
$
```

习题14 Windows会话

```
> cd temp
```

> ls

Directory: C:\Users\zed\temp

Mode		LastWriteTime	Length	Name
----		-----	-----	----
d----	12/22/2011	4:52 PM		newplace
d----	12/22/2011	4:52 PM		something
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt
-a---	12/22/2011	4:49 PM	0	thefourthfile.txt
-a---	12/22/2011	4:49 PM	0	uncool.txt

> rm uncool.txt

> ls

Directory: C:\Users\zed\temp

Mode		LastWriteTime	Length	Name
----		-----	-----	----
d----	12/22/2011	4:52 PM		newplace
d----	12/22/2011	4:52 PM		something
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt
-a---	12/22/2011	4:49 PM	0	thefourthfile.txt

> rm iamcool.txt

> rm neat.txt

> rm thefourthfile.txt

> ls

Directory: C:\Users\zed\temp

Mode		LastWriteTime	Length	Name
----		-----	-----	----

```
d----      12/22/2011   4:52 PM           newplace
d----      12/22/2011   4:52 PM           something
> cp -r something newplace
> rm something/awesome.txt
> rmdir something
> rm -r newplace
> ls
>
```

知识点

这里我们将上一个习题中的文件清理掉了。先前我让你用`rmdir`来删除包含文件的目录，但是操作失败了，失败的原因是你不能用这条命令删除包含文件的目录。要移除这样的目录，需要先删除文件，或者循环删除目录下的所有内容，这也是这个习题的结尾所做的事情。

更多任务

删除目录下至今为止的所有内容。

在笔记本上记下来：循环移除文件时要小心操作。

习题15 退出命令行 (**exit**)

任务

习题15 Linux/Mac OSX会话

```
$ exit
```

习题15 Windows会话

```
> exit
```

知识点

最后一个练习是如何退出你的命令行终端。这本身很简单，不过我还有一些额外的任务给你。

更多任务

作为最后的练习，我将要求你通过帮助系统来学习并使用一些命令。

Unix的命令列表：

xargs

sudo

chmod

chown

Windows的目录列表：

forfiles

runas

attrib

icacls

学习它们的用途，试着使用它们，再把它们加到你的索引卡中。

命令行将来的路

你已经读完这个快速入门。到这里，你的水平基本上达到了能使用shell的程度了。其实要学的技巧和命令用法还有很多，这里我会再给你一些阅读和研究方向。

Unix Bash参考资料

你一直在用的shell叫做Bash。它不见得是最牛的shell，不过你到处都能看到它，而且它也有不少的功能，所以作为入门是很不错的。接下来我给你提供一些关于Bash的链接，好好阅读一下。

Bash

Cheat

Sheet:

http://cli.learncodethehardway.org/bash_cheat_sheet.pdf，作者Raphael，CC license。

Reference

Manual:

<http://www.gnu.org/software/bash/manual/bashref.html>。

PowerShell参考资料

Windows下能用的也就只有PowerShell了。下面是关于PowerShell的一些有用的链接。

Owner's

Manual: <http://technet.microsoft.com/en-us/library/ee221100.aspx>。

Cheat

Sheet: <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=7097>。

Master

Powershell:

<http://powershell.com/cs/blogs/ebook/default.aspx>。